(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) EP 1 669 885 A2

(12) EUROPÄISCHE PATENTANMELDUNG

(43) Veröffentlichungstag:
14.06.2006 Patentblatt 2006/24

(51) Int Cl.:
G06F 15/76 *(2006.01)*

(21) Anmeldenummer: **05008631.3**

(22) Anmeldetag: **07.02.1998**

(84) Benannte Vertragsstaaten:
AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC NL PT SE

(30) Priorität: **08.02.1997 DE 19704728**

(62) Dokumentnummer(n) der früheren Anmeldung(en) nach Art. 76 EPÜ:
98909346.3 / 0 961 980

(71) Anmelder: **PACT XPP Technologies AG**
80939 München (DE)

(72) Erfinder:
- **Vorbach, Martin**
  67360 Lingenfeld (DE)
- **Münch, Robert**
  76135 Karlsruhe (DE)

(74) Vertreter: **Pietruk, Claus Peter**
Heinrich-Lilienfein-Weg 5
76229 Karlsruhe (DE)

Bemerkungen:
Diese Anmeldung ist am 20 - 04 - 2005 als Teilanmeldung zu der unter INID-Kode 62 erwähnten Anmeldung eingereicht worden.

(54) Verfahren zur Selbstsynchronisation von konfigurierbaren Elementen eines programmierbaren Bausteines

(57) Die Erfindung betrifft ein Verfahren zur Synchronisation der Datenverarbeitung zur Laufzeit konfigurierbarer, datenverarbeitender Elemente einer zur Laufzeit rekonfigurierbaren Zellarchitektur mit den Schritten, dass zumindest einige der Elemente konfiguriert werden, ein in einer Zelle erzeugtes Triggersignal zur Laufzeit zu einer Anzahl konfigurierter Elemente innerhalb der Zellarchitektur übertragen wird, das Triggersignal von der Anzahl der Elemente empfangen wird und im Ansprechen auf den Triggerempfang anhand dessen die Ausführung einer Konfiguration zur Datenverarbeitung bestimmt wird

EP 1 669 885 A2

Beschreibung

1. Hintergrund der Erfindung

1.1 Stand der Technik

1.1.1 Probleme

[0001] Bei heutigen Bausteinen (FPGA, DPGA etc.) wird die Synchronisation der konfigurierbaren Elemente meistens durch den Takt des Bausteines hergestellt. Diese Art der zeitlich gesteuerten Synchronisation bereitet viele Probleme, da oft nicht im Vorhinein bekannt ist, wie lange eine Aufgabe benötigt, bis ein gültiges Ergebnis bereit steht. Ein weiteres Problem der zeitgesteuerten Synchronisation ist, daß das Ereignis auf welches die Synchronisation erfolgt nicht von dem zu synchronisierenden Element selbst ausgelöst wird, sondern von einem unabhängigem Element. In diesem Fall sind nun zwei verschiedene Elemente an der Synchronisation beteiligt. Dies führt zu einem erheblich höhrem Verwaltungsaufwand.
Aus EP-A-0,726,532 ist ein Verfahren zur Steuerung des Datenflußes in, aus mehreren als Array angeordneten Prozessoren bestehenden, SIMD-Maschinen bekannt. Dabei wird eine Instruktion an alle Prozessoren gesendet, die dynamisch den Zielprozessor einer Datenübertragung auswählt. Die Instruktion wird von einer übergeordneten Instanz an alle Prozessoren (broadcast instruction)gesendet und besteht aus einem Richtungsfeld (destination field) und einem Zielfeld (target field). Das Richtungsfeld steuert eine Einheit im Prozessorelement, um dynamisch das Nachbar-Prozessorelement zu ermitteln, zu dem das Ergebnis gesendet werden soll. Mit dem Zielfeld wird dynamisch das Operandenregister eines weiteren Prozessorelements ausgewählt, in welchem ein weiteres Ergebnis gespeichert werden soll.

1.1.2 Verbesserung durch die Erfindung

[0002] Durch die Erfindung wird ein Verfahren beschrieben, welches es gestattet, daß die Synchronisation von zu synchronisierenden Elementen selbst ausgeht. Die Synchronisation ist nicht mehr durch eine zentrale Instanz implementiert und wird auch nicht mehr durch eine zentrale Instanz verwaltet. Durch die Verlegung der Synchronisation in jedes Element können auch viel mehr Synchronisationsaufgaben gleichzeitig durchgeführt werden, da unabhängige Elemente sich nicht mehr gegenseitig beim Zugriff auf die zentrale Synchronisations-Instanz behindern. Die Einzelheiten und besondere Ausgestaltungen, sowie Merkmale des erfindungsgemäßen Synchronisationsverfahrens sind Gegenstand der Patentansprüche.

2. Beschreibung der Erfindung

'2.1 Übersicht über die Erfindung, Abstrakt

[0003] In einem Baustein (DFP, DPGA) mit zwei- oder mehrdimensional angeordneter, programmierbarer Zellstruktur kann jedes konfigurierbare Element über eine Vernetzungsstruktur auf die Konfigurations- und Statusregister der anderen konfigurierbaren Elemente zugreifen und damit deren Funktion und Arbeitsweise aktiv beeinflussen. Eine Matrix aus derartigen Zellen wird im Folgenden ProcessingArray (PA) genannt. Die Konfiguration kann somit zusätzlich zu der üblichen Methode durch eine Ladelogik, aus dem ProcessingArray (PA) heraus erfolgen.

2.2 Detailbeschreibung der Erfindung

[0004] Es wird von einem frei zur Laufzeit programmierbaren Baustein ausgegangen, welcher zusätzlich zur Laufzeit rekonfiguriert werden kann. Die auf dem Chip enthaltenen konfigurierbaren Elemente besitzen ein oder mehrere Konfigurationsregister für verschiedene Aufgaben. Auf diese Konfigurationsregister kann lesend wie schreibend zugegriffen werden. In dem beschriebenen Verfahren wird davon ausgegangen, daß für folgende Informationen eine Konfiguration in einem zu konfigurierenden Element eingestellt werden kann.

- Vernetzungs-Register. In diesem Register wird die Art der Verbindung zu anderen Zellen eingestellt.
- Befehls-Register. In diesem Register wird die auszuführende Funktion des konfigurierbaren Elements eingetragen.
- Status-Register. In diesem Register speichert die Zelle ihren aktuellen Zustand. Dieser Zustand gibt anderen Elementen des Bausteins Auskunft darüber, in welchem Verarbeitungszyklus sich die Zelle befindet.

Eine Zelle wird durch einen Befehl konfiguriert, welcher die Funktion der Zelle bestimmt, die ausgeführt werden soll. Weiterhin werden Konfigurationsdaten eingetragen um die Vernetzung mit anderen Zellen und den Inhalt des Status-

Registers einzustellen. Nach diesem Vorgang ist die Zelle betriebsbereit.

[0005] Um eine flexible und dynamische Zusammenarbeit vieler Zellen zu ermöglichen, kann jede Zelle auf alle Konfigurationsregister einer anderen Zelle lesend oder schreibend zugreifen. Auf welches der vielen Konfigurationsregister lesend oder schreibend zugegriffen wird, wird durch die Art des Befehls, mit welchem die Zelle konfiguriert wurde, festgelegt. Jeder Befehl den die Zelle ausführen kann, existiert in soviel verschiedenen Adressierungsarten, wie es verschiedene, voneinander unabhängige Konfigurationsregister, in einem zu konfigurierenden Element gibt.

[0006] Beispiel: Eine Zelle besitzt die oben angegebenen Konfigurationsregister (Vernetzung, Befehl und Status) und soll den Befehl ADD, welcher eine Addition durchführt ausführen. Durch die verschiedenen Arten des ADD Befehls kann nun selektiert werden, wohin das ergebnis dieser Funktion übertragen wird.

ADD-A. Das Ergebnis wird an das Operand-Register-A der Zielzelle übertragen.
ADD-B. Das Ergebnis wird an das Operand-Register-B der Zielzelle übertragen.
ADD-V. Das Ergebnis wird an das Vernetzungs-Register der Zielzelle übertragen.
ADD-S. Das Ergebnis wird an das Status-Register der Zielzelle übertragen.
ADD-C. Das Ergebnis wird an das Befehls-Register der Zielzelle übertragen.

2.2.1 Steuer- und Synchonisations Trigger

[0007] Neben dem Ergebnis kann jede Zelle eine Menge an TriggerSignalen erzeugen. Die Trigger-Signale müssen nicht notwendigerWeise an die gleiche Zielzelle übertragen werden, wie das Ergebnis der Verarbeitung des konfigurierten Befehles. Ein Trigger-Signal oder erst die Kombination mehrerer Trigger-Signale, löst bei der Zielzelle eine bestimmte Aktion aus oder setzt die Zelle in einen bestimmten Zustand. Eine Beschreibung der Zustände ist weiter unten im Text zu finden. Folgende Trigger-Signale gibt es:

- GO-Trigger. Der GO-Trigger setzt die Zielzelle in den Zustand READY.
- RECONFIG-Trigger. Der RECONFIG-Trigger setzt die Zielzelle in den Zustand RECONFIG, so daß die Zelle umprogrammiert werden kann. Besonders in Zusammenarbeit mit Switching-Tabellen ist dieser Trigger sehr sinnvoll. Geht man davon aus, daß zu verarbeitenden Daten mit der steigenden Taktflanke in die Operanden-Register geladen werden, in der Zeitspanne des H-Level verarbeitet werden und mit der fallenden Flanke in das Ausgangsregister geschrieben werden, so ist eine Rekonfigurierung der Zelle mit der fallenden Flanke möglich. Mit der fallenden Flanke werden die neuen Konfigurationsdaten in das Befehls-Register geschrieben. Die Zeitspanne des L-Level ist ausreichend genug, um die Rekonfigurierung erfolgreich abzuschließen.
- STEP-Trigger. Der STEP-Trigger löst bei der Zielzelle, welche sich im Zustand WAIT befindet, die einmalige Ausführung des konfigurierten Befehls aus.
- STOP-Trigger. Der STOP-Trigger hält die Zielzelle an, in dem die Zelle in den Zustand STOP gesetzt wird.

[0008] Durch die Möglichkeit in der verarbeitenden Zelle anzugeben, in welches Register der Zielzelle das Ergebnis eingetragen werden soll und welche Art von Trigger-Signal erzeugt werden soll, kann aus einem Datenstrom eine Menge an Verwaltungsdaten erzeugt werden. Diese Verwaltungsdaten stellen kein Ergebnis der eigentlichen Aufgabe dar, welche durch den Chip abgearbeitet werden soll, sondern dienen allein der Verwaltung, Synchronistion, Optimierung etc. des internen Zustands.

[0009] Jede Zelle kann folgende Zustände annehmen, welche durch eine geeignete Kodierung im Status-Register dargestellt werden.

- READY. Die Zelle ist mit einem gültigen Befehl konfiguriert worden und kann Daten verarbeiten. Die Verarbeitung findet mit jedem Taktzyklus statt. Die Daten werden auf Grund der Adressierungsart der datenschickenden Zelle in die Register der Zielzelle eingelesen.
- WAIT. Die Zelle ist mit einem gültigen Befehl konfiguriert worden und kann Daten verarbeiten. Die Verarbeitung findet mit auf Grund eines Trigger-Signals statt, welches durch andere Elemente des Bausteins erzeugt werden können. Die Daten werden auf Grund der Adressierungsart der datenschickenden Zelle in die Register der Zielzelle eingelesen.
- CONFIG. Die Zelle ist nicht mit einem gültigen Befehl konfiguriert. Das Datenpaket, welches mit dem nächsten Taktzyklus an die Zelle gesandt wird, wird in das Befehls-Register eingelesen. Das Datenpaket wird auf jeden Fall in das Befehls-Register eingelesen, egal welche Adressierungsart von der datenschickenden Zelle benutzt wurde.
- CONFIG-WAIT. Die Zelle ist nicht mit einem gültigen Befehl konfiguriert. Ein Datenpaket, wird mit dem nächsten Trigger-Signal, welches durch andere Elemente des Bausteins erzeugt werden kann, eingelesen und in das Befehls-Register geschrieben. Das Datenpaket wird auf jeden Fall in das Befehls-Register eingelesen, egal welche Adressierungsart von der datenschickenden Zelle benutzt wurde.

- RECONFIG. Die Zelle ist mit einem gültigen Befehl konfiguriert, verarbeitet aber keine weiteren Daten, nimmt die Daten auch nicht an. Die Zelle kann durch ein anderes Element des Bausteins umkonfiguriert werden.
- STOP. Die Zelle ist mit einem gültigen Befehl konfiguriert, verarbeitet aber momentan keine Daten. Die Daten werden von der Zelle angenommen (in die Eingangsregister übertragen), aber nicht weiterverarbeitet.

[0010] Durch diese verschiedenen Zustände und der Möglichkeit auf die verschiedene Register einer Zelle schreibend und lesend zuzugreifen, kann jede Zelle eine aktive Verwaltungsrolle einnehmen. Im Gegensatz dazu besitzen alle existierenden Bausteine dieser Art eine zentrale Verwaltungsinstanz, welche immer den gesamten Zustand des Bausteins kennen und handhaben muß.

[0011] Um eine weitere Flexibilität zu erreichen gibt es eine weitere Klasse an Befehlen, die nach der ersten Ausführung ihre Art wechseln. Bezogen auf des Beispiel des ADD-Befehls sieht ein Befehl dann so aus:

- ADD-C-A. Das Ergebnis der ADD Funktion wird bei der ersten Ausführung des Befehls in das Befehls-Register der Zielzelle geschrieben. Bei jeder weiteren Ausführung wird das Ergebnis in das Operand-Register-A geschrieben.

[0012] Diese Möglichkeit kann beliebig erweitert werden, so daß auch Befehle der Art ADD-C-V-A-C-...-B denkbar sind. Jeder Befehl kann alle permutierten Kombinationen der verschiedenen Adressierungs- und Trigger-Arten annehmen.

2.2.2 Rekonfigurationssteuerung mittels RECONFIG-Trigger:

[0013] Im bisherigen Verfahren war es notwendig, daß jedes zu konfigurierendes Element von einer externen Instanz einen RECONFIG-Trigger erhalten mußte, um in den Zustand 'rekonfigurierbar' überzugehen. Dies hat den Nachteil, daß für die Verteilung des RECONFIG-Triggers ein erheblicher Vernetzungsaufwand und Konfigurationsaufwand erforderlich war.

[0014] Durch die Strukutur der Vernetzung, kann dieser Nachteil beseitigt werden. Alle konfigurierbaren Elemente, welche durch die Vernetzungsinformation zusammenhängen, stellen einen gerichteten Graphen dar. Ein solcher Graph kann mehrere Wurzeln (Quellen) und mehrere Blätter (Ziele) haben. Die konfigurierbaren Elemente werden so erweitert, daß sie einen eingehenden RECONFIG-Trigger entweder in Richtung ihrer ausgehenden Register, eingehenden Register, oder einer Kombination derer, propagieren. Durch diese Propagierung erhalten alle direkt mit dem konfigurierbaren Element verbundenen konfigurierbaren Elemente ebenfalls den RECONFIG-Trigger.

[0015] Eine Konfiguration (Graph) kann nun komplett in den Zustand 'rekonfigurierbar' gebracht werden, in dem entweder an alle Wurzeln ein RECONFIG-Trigger geschickt wird und diese den RECONFIG-Trigger in Richtung der Ausgangsregister propagieren. Die Menge der Wurzeln in einem Graph, an die ein RECONFIG-Trigger geschickt werden muß, ist erheblich kleiner, als die Menger aller Knoten des Graphen. Dadurch wird eine erhebliche Minimierung des Aufwandes erreicht. Selbstverständlich kann ein RECONFIG-Trigger auch an alle Blätter geschickt werden. Der RECONFIG-Trigger wird in diesem Fall in Richtung der Eingangsregister propagiert.

[0016] Durch den Einsatz beider Möglichkeiten oder einer Mischung beider Verfahren, kann die minimale Menge an konfigurierbaren Elementen berechnet werden, an die ein RECONFIG-Trigger herangeführt werden muß.

[0017] Die konfigurierbaren Elemente können einen Zusatz zu ihrem Status-Register bekommen, der angibt, ob ein eingehender RECONFIG-Trigger propagiert werden soll oder nicht. Diese Information wird dann benötigt, wenn zwei oder mehr verschiedene Graphen an ein oder mehreren Stellen zusammen hängen (also einen Übergang haben) und es nicht gewünscht ist, daß auch einer der anderen Graphen in den Zustand 'rekonfigurierbar' übergeht. Ein oder mehrere konfigurierbare Elemente verhalten sich also wie eine Schleuse.

[0018] Weiterhin kann das Status-Register derart erweitert werden, daß ein zusätzlicher Eintrag angibt, in welche Richtung ein eingehender RECONFIG-Trigger weitergegeben werden soll.

[0019] Das beschriebene Verfahren, kann auf alle Arten von Triggern und/oder Daten angewandt werden. Es kann dadurch eine automatische
Verteilungshierarchie hergestellt werden, welche sehr wenige Zugriffsmöglichkeiten von außen benötigt, um diese in Gang zu setzen.

3. Implementierung mehrer Funktionen gleichzeitig in denselben konfigurierbaren Elementen

3.1 Grundfunktion und benötigte Trigger

[0020] Eine besonders komplexe Variante des Aufrufes verschiedener Macros durch eine Bedingung wird im Folgenden vorgestellt:

Bei der Ausführung einer Bedingung (IF *COMP* THEN A ELSE B; wobei COMP ein Vergleich darstellt, A und B auszuführende Operationen sind) werden keine GO- und STOP-Trigger generiert. Statt dessen wird ein Triggervektor (TRIGV) generiert, der angibt, zu welchem Ergebnis der Vergleich COMP geführt hat. Der Triggervektor kann daher die Zustände "gleich", "größer" oder "kleiner" annehmen.
Der Vektor wird an eine nachfolgende Zelle gesandt, die anhand des Zustandes des Vektors genau ein bestimmtes Konfigurationsregister (entsprechend A oder B) aus einer Mehrzahl von Konfigurationsregistern auswählt. Dadurch wird erreicht, daß je nach Ergebnis des vorangegangenen Vergleiches eine andere Funktion über die Daten durchgeführt wird.
Zustände wie "größer-gleich", "kleiner-gleich", "gleichungleich" werden aufgelöst, indem zwei Konfigurationsregister mit denselben Konfigurationsdaten beschrieben werden. Beispielsweise wird bei "größer-gleich" das Konfigurationsregister "größer" und das Konfigurationsregister "gleich" mit demselben Konfigurationswort beschrieben, während das Konfigurationsregister "kleiner" ein anderes Konfigurationswort enthält.

[0021] Bei der Implementierung der Triggervektoren (TRIGV) ist keine Begrenzung auf die Zustände "größer", "kleiner" und "gleich" erforderlich. Zur Auswertung von großen "CASE ... OF" Konstrukten kann eine beliebige Zahl n, die den Zustand des CASE repräsentiert, als Triggervektor (TRIGV-m) zu der oder den nachfolgen Zellen weitergeleitet werden. Mit anderen Worten gibt n den Vergleich innerhalb des CASE an, der bei Auswertung der anliegenden Daten zugetroffen hat. Zur Ausführung der innerhalb des CASE dem Vergleich zugeordneten Funktion wird n an die ausführenden Zellen weitergeleitet um die entsprechende Funktion zu selektieren. Während die Zellen im "größer/kleiner/gleich"-Fall mindestens 3 Konfigurationsregister benötigen, muß bei der Verwendung von TRIGV-m die Anzahl der Konfigurationsregister mindestens genau dem maximalen Wert von n (max (n)) entsprechen.

3.2 Propagieren der benötigten Funktion durch Trigger

[0022] TRIGV/TRIGV-m werden an die erste, die Daten verarbeitende Zelle gesendet. In dieser Zelle werden TRIGV/TRIGV-M ausgewertet und die Daten entsprechend verarbeitet. Zusammen mit den Daten wird TRIGV/TRIGV-m an die nachfolgenden Zellen weitergeleitet (propagiert). Dabei erfolgt eine Weiterleitung an alle Zellen, die aufgrund der Auswertung (IF oder CASE) eine bestimmte Funktion ausführen. Dabei ist die Weiterleitung direkt an die Weiterleitung der Datenpakete gekoppelt, d.h die Weiterleitung erfolgt synchron zu den Daten. Die zum Zeitpunkt t generierten TRIGV/TRIGV-m, werden mit den zum Zeitpunkt t an den ersten verarbeitenden Zellen (CELLS1, vgl. Fig. 5: 0502, 0505, 0507) anstehenden Daten verknüpft.

[0023] TRIG/TRIG-V werden so weitergeleitet, daß die Vektoren mit den Daten zum Zeitpunkt t+1 an den zweiten verarbeitenden Zellen anliegen und zum Zeitpunkt t+2 an den dritten verarbeitenden Zellen usw. bis zum Zeitpunkt t+m TRIG/TRIG-V und die Daten an den (m-1)-ten Zellen und gleichzeitig letzten Zellen, die von dem TRIG/TRIG-V auslösendem Vergleich (IF/CASE) abhängig sind, anstehen. Eine Verknüpfung erfolgt keinesfalls so, daß die zum Zeitpunkt t generierten TRIG/TRIG-V mit Daten verknüpft werden, die zu einem Zeitpunkt $t_{old}<t$ an CELLS1 anlagen!

3.3 Reagieren auf das Auftreten oder Nicht-Auftreten von Triggern

[0024] Es ist in Sonderfällen erforderlich auf das Nicht-Vorhandensein eines Triggers zu reagieren, d.h. ein Triggerzustand tritt auf, jedoch wird keine Änderung des Triggervektors ausgelöst. Auch in diesem Fall kann eine sinnvolle und wichtige Information an die nachfolgenden Zellen übertragen werden. Beispielsweise ist bei einem Vergleich auf "größer", "kleiner", "gleich" das Triggersignal "gleich" nicht vorhanden und ändert sich auch nicht, wenn vom Zustand "kleiner" zum Zustand "größer" übergegangen wird. Dennoch beinhaltet das Nicht-Vorhandensein von "gleich" eine Information, nämlich "ungleich".
Um auf beide Zustände "vorhanden" und "nicht vorhanden" reagieren zu können, wird ein Eintrag in das Konfigurationsregister der Zelle hinzugefügt, das angibt, auf welchen der Zustände reagiert werden soll.
Zudem wird zum Triggervektor TRIGV, der die Zustände "gleich", "größer" und "kleiner" repräsentiert, ein Signal TRIGRDY hinzugefügt, der das Auftreten eines Triggers anzeigt. Dies ist notwendig, da der Zustand "nicht vorhanden" auf einem der Vektoren keinen Aufschluß mehr über das Vorhandensein eines Trigger an sich gibt.
TRIGRDY kann für ein Handshaking-Protokoll zwischen der sendenen und empfangenden Zelle benutzt werden, indem die empfangende Zelle einen TRIGACK generiert, sobald sie die Triggervektoren ausgewertet hat. Erst nach Eintreffen des TRIGACK nimmt die sendende Zelle den Triggerzustand zurück.
Dabei wird anhand eines Eintrages in das Konfigurationsregisters festgelegt, ob bei Aussenden eines Triggervektors auf den Erhalt eines TRIGACK gewartet werden soll, oder ob der Triggerkanal unsynchronisiert abläuft.

3.4 Einsatz in Mikroprozessoren

[0025] In Mikroprozessoren neuester Architektur werden bedingte Sprünge nicht mehr nach dem bekannten Verfahren der Branch-Prediction, also der Vorhersage eines Sprunges ausgeführt. Die spekulative Vorhersage von Sprüngen, die zur Leistungssteigerung von Prozessoren eingeführt wurde, berechnete Sprünge aufgrund von spekulativen Algorithmen voraus und mußte bei fehlerhaften Berechnungen die gesamte Prozessorpipeline neu laden, was zu erheblichen Leistungsverlusten führte. Um diese Verluste zu eleminieren wurde das neue Predicate/NOP-Verfahren eingeführt. Dabei ist jedem Befehl ein ein Bit breites Status-Flag zugeordnet, das anzeigt, ob der Befehl ausgeführt werden soll - oder nicht. Dabei kann eine beliebige Menge an Statusflags existieren. Die Zuordnung von Befehlen zu Status-Flags geschieht durch einen Compiler während der Übersetzung des Codes. Die Status-Flags werden von den ihnen zugeordneten Vergleichsoperationen zur Ausführungzeit verwaltet und zeigen das Ergebnis des jeweiligen Vergleiches an.

[0026] Je nach Zustand des einem Befehl zugeordneten Status-Flag wir der Befehl dann vom Prozessor ausgeführt (sofern das Status-Flag "ausführen" anzeigt) oder der Befehl wird nicht ausgeführt und durch einen NOP ersetzt (sofern das Status-Flag "nicht ausführen" anzeigt). Ein NOP steht für "No OPeration", was bedeutet, daß der Prozessor in diesem Zyklus keine Operation ausführt. Dadurch geht der Zyklus für sinnvolle Operationen verloren.
Zur Optimierung des Zyklusverlustes werden zwei Möglichkeiten vorgeschlagen:

3.5.1 Mehrere Befehlsregister pro Recheneinheit

[0027] Eine moderner Mikroprozessor besitzt mehrere relativ unabhängige Rechenwerke.
Gemäß dem hier vorgestellten Trigger-Prinzip können die einzelnen Recheneinheiten mit mehreren Befehlsregistern ausgestattet werden, wobei ein Befehlsregister eines Mikroprozessorrechenwerkes synonym für ein Konfigurationsregister, gemäß üblichen FPGA, DFP, o.ä. Bausteinen, steht. Die Auswahl des jeweilig aktiven Befehlsregisters erfolgt

a) anhand von Triggervektoren, die anderen Rechenwerken anhand von Vergleichen generiert wurden.
b) anhand von mehrbittigen Status-Flags (im folgenden Status-Vektoren genannt), die gemäß dem heutigen Verfahren nach dem Stand der Technik, Vergleichsbefehlen zugeordnet sind.

3.5.2 Geänderter VLIW-Befehlssatz

[0028] Eine besondere Ausgestaltung bietet sich durch VLIW-Befehlssätze. So kann innerhalb eines Befehlswortes mehrere möglichen, von einem Vergleich abhängenden, Befehle zu einem Befehl zusammengefaßt werden. Ein VLIW-Wort beliebiger Breite wird in eine beliebige Menge an Befehlen (Codes) unterteilt.

[0029] Jeder einzelne dieser Codes wird durch einen Triggervektor oder Status-Vektor referenziert. Das bedeutet, zur Laufzeit wird einer der vorhandenen Codes aus dem VLIW-Wort ausgewählt und verarbeitet.

[0030] In der Tabelle ist ein mögliches VLIW-Wort mit vier Codes abgebildet, auf das ein 2-bittiger Triggervektor oder ein 2-bittiges Statusflag referenziert:

VLIW-Befehlswort:	Code0	Code1	Code2	Code3

Zuordnung:

Triggervektor / Status-Flag	00	01	10	11

4. Erweiterung der Hardware gegenüber üblichen FPGAs und DFPs

4.1 Zusätzliche Register

[0031] Zu den in DFPs üblichen Konfigurationsregistern kommt ein Statusregister und ein Konfigurationsregister hinzu. Belde Register werden vom PLU-Bus angesteuert und haben Verbindung zur Zustandsmaschine der Ablaufsteuerung der jeweiligen Zelle.

4.2 Veränderung des PLU-Busses

[0032] In FPGAs und DFPs werden die konfigurierbaren Register M-/F-PLUREG ausschließlich über den PLU-Bus, der die Verbindung zur Ladelogik darstellt, verwaltet. Um die erfindungsgemäße Funktion zu gewährleisten muß nunmehr eine zusätzliche Zugriffsmöglichkeit durch den normalen Systembus zwischen den Zellen möglich sein. Dasselbe gilt für die neuen Status- und Konfigurationsregister.

[0033] Dabei ist nur der Teil des Systembusses für die Register relevant, der über die BM-UNIT, also dem Interface zwischen den Systembusse und der PAE, mit der PAE vernetzt ist.
Daher wird der Bus von der BM-UNIT an die Register weitergeleitet, wo vorgeschaltete Multiplexer oder vorgeschaltete Tore die Umschaltung zwischen dem PLU-Bus und dem für die PAE relevanten Systembus übernehmen.
Dabei sind die Multiplexer oder Tore so geschaltet, daß sie immer den für die PAE relevanten Systembus durchschalten, außer nach einem Rücksetzen des Bausteines (RESET) oder wenn der RECONFIG-Trigger aktiv ist.

4.3 Erweiterungen der konfigurierbaren Elemente (PAEs) gegenüber üblichen FPGAs und DFPs

4.3.1 Triggerquellen

[0034] Ein konfigurierbares Element kann Trigger von mehreren Quellen gleichzeitig empfangen. Durch diese Möglichkeit kann mit Hilfe von Maskierungs-Registern eine flexiblere Semantik der Trigger erreicht werden.

4.3.2 Mehrere Konfigurationsregister

[0035] Anstatt eines Konfigurationsregisters besitzt eine PAE mehrere (max(n)) Konfigurationsregister.

4.3.3 Konfigurationsstatemachine und Multiplexer

[0036] Den Konfigurationsregistern nachgeschaltet ist ein Multiplexer, der eine der möglichen Konfigurationen auswählt. Die Steuerung des Multiplexers erfolgt durch eine separate oder in die PAE-Statemachine integriert Statemachine, die den Multiplexer anhand eingehender Triggervektoren steuert.

'4.3.4 Triggerauswertung und Konfiguration

[0037] Ein konfigurierbares Element kann ein Maskierungs-Register enthalten, in dem eingestellt werden kann, auf welchen Trigger-Eingängen, ein
Trigger-Signal anliegen muß, so daß die Bedingungen für eine Aktion des konfigurierbaren Elements, erfüllt sind. Ein konfigurierbares Element reagiert nicht nur auf einen Trigger, sondern auf eine eingestellte Kombination aus Triggern. Weiterhin kann ein konfigurierbares Element eine Priorisierung gleichzeitig eingehender Trigger vornehmen.
Eingehende Trigger werden anhand des TRIGRDY-Signals erkannt. Dabei werden die Triggervektoren gemäß zusätzlich in den Konfigurationsregistern vorhandenen Konfigurationsdaten ausgewertet.

4.3.5 Triggerhandshake

[0038] Sobald die Triggervektoren ausgewertet sind, wird ein TRIGACK zur Bestätigung des Triggervektors generiert.

4.3.6 BM-UNIT

[0039] Die BM-Unit wird so erweitert, daß sie vom Bus kommende Trigger gemäß der Konfiguration im M-PLUREG an die Sync-Unit und SM-Unit weiterreicht. Von der EALU generierte Trigger (z.B. Vergleicherwerte "größer", "kleiner", "gleich", 0-Detektoren, Vorzeichen, Überträge, Fehlerzustände (Division durch 0, etc.), etc.) werden gemäß der Verschaltungsinformation im M-PLUREG von der BM-UNIT an den Bus weitergeleitet.

4.4 Erweiterungen des Systembusses

[0040] Der Systembus, also das Bussystem zwischen den Zellen (PAEs), wird dahingehend erweitert, daß zusammen mit den Daten die Informationen über die Zielregister übertragen werden. Das bedeutet, eine Adresse wird mitgeschickt, die beim Datenempfänger das gewünschte Register selektiert.
Ebenfalls wird der Systembus um die unabhängige Übertragung von Triggervektoren und -handshakes erweitert.

5. Kurzbeschreibung der Diagramme

[0041]

Fig. 1 zeigt, wie durch den Einsatz von Triggern ein Schleifenkonstrukt implementiert werden kann.
Fig. 2 zeigt, wie durch den Einsatz mehrerer Trigger ein Vergleichskonstrukt implementiert werden kann.
Fig. 3 zeigt, wie durch den Einsatz mehrerer Trigger und deren Verschachtelung ein Vergleichskonstrukt mit mehreren Ausgängen implementiert werden kann.
Fig. 4 zeigt die notwendigen Erweiterungen gegenüber üblichen FPGAs und DFPs.
Fig. 5 zeigt in einem Funktionsbeispiel die Auswahl verschiedener Funktionen der konfigurierbaren Elemente durch Trigger.
Fig. 6 zeigt die Implementierung von mehreren durch Trigger angesteuerten Konfigurationsregistern zur Ausführung verschiedener Funktionen.
Fig. 7 zeigt die Implementierung des Verfahrens aus Fig. 6 in Mikroprozessoren

6. Detailbeschreibung der Diagramme und Ausführungsbeispiele

[0042]

Figur 1
Das Makro 0103 soll in diesem Beispiel 70 mal ausgeführt werden. Eine Ausführung des Makros benötigt 26 Taktzyklen. Das bedeutet, daß nur alle 26 Taktzyklen der Zähler 0101 um eins verringert werden darf. Ein Problem bei frei programmierbaren Bausteinen ist nun, daß nicht immer garantiert werden kann, daß auch wirklich nach 26 Takten die Abarbeitung des Makros 0103 abgeschlossen ist. Eine Verzögerung kann zum Beispiel dadurch entstehen, daß ein Makro, welches die Eingangsdaten für Makro 0103 liefern soll, plötzlich 10 Taktzyklen länger benötigt. Aus diesem Grund sendet die Zelle in Makro 0103 ein Trigger Signal an den Zähler 0101, durch welche das Ergebnis der Berechnung an ein weiteres Makro gesandt wird. Gleichzeitig wird die Verarbeitung des Makros 0103 durch die gleiche Zelle gestoppt. Diese Zelle 'weis' genau, daß die Bedingung für die Beendigung einer Berechnung erreicht wurde.
Das gesendete Trigger-Signal ist in diesem Fall ein STEP-Trigger, welcher veranlaßt, daß der Zähler 0101 einmal seine konfigurierte Funktion ausführt. Der Zähler zählt seinen Zählerwert um eine herunter und vergleicht, ob er den Wert 0 erreicht hat. Ist dies nicht der Fall, wird ein GO-Trigger an das Makro 0103 abgeschickt. Dieses GO-Trigger-Signal veranlaßt das Makro 0103 seine Funktion wieder aufzunehmen.
Dieser Vorgang wiederholt sich solange, bis der Zähler 0101 den Wert 0 erreicht hat. In diesem Fall wird ein Trigger-Signal an das Makro 0102 geschickt und löst dort eine Funktion aus.
Durch dieses Zusammenspiel von Triggern kann eine sehr feingranulare Synchronisation erreicht werden.
Figur 2
Figur 2 entspricht der Grundidee her Figur 1. Die Funktion in Element 0202 ist diesesmal jedoch kein Zähler sondern ein Vergleicher. Das Makro 0201 schickt nach jedem Verarbeitungsdurchlauf einen Verleichswert mit an den Vergleicher 0202. Je nach Ausgang des Vergleichs, werden wiederum verschiedene Trigger angesteuert um zum Beispiel eine Aktion in den Makros 0203 zu veranlassen. Das in Figur 2 implementierte Konstrukt entspricht dem einer IF-Abfrage in einer Progammiersprache.
Figur 3
Wie in Figur 2 werden hier mehrere Vergleicher 0301, 0302 eingesetzt, um die Konstruktion eines IF-ELSE-ELSE Konstruktes (oder einer Mehrfachauswahl) zu implementieren. Durch die Verwendung verschiedenster Arten von Triggern und Verbindungen dieser Trigger zu den Makros 0303, 0304 können sehr komplexe Abläufe einfach implementiert werden.
Figur 4
zeigt die Unterschiede zu üblichen FPGAs und DFPs. Das hinzugefügte Konfigurationsregister (0401) und das hinzugefügte Statusregister (0402) haben über den Bus (0407) Verbindung zur SM-UNIT. Die Register 0401, 0402, F- und M-PLUREG sind über einen internen Bus 0206 mit einem Tor 0403 verbunden. Dieses verbindet den internen

Bus (0406) je nach Stellung mit dem PLU-Bus 0405 um eine Konfiguration durch die PLU zu ermöglichen oder über einen Bus 0408 mit dem BM-UNIT. Diese schaltet je nach Adressierung auf dem Datenbus 0404 die Daten zu den O-REG weiter oder zu dem adressierten Register 0401, 0402, F- oder M-PLUREG.

Die BM-UNIT (0411) sendet über 0415 Triggersignale an die SYNC-UNIT (0412). Von der EALU erhält 0411 über 0414 Ergebnisse ("gleich", "größer", "kleiner", "Ergebnis = 0", "Ergebnis positiv", "Ergebnis negativ", Überlauf (positiv und negativ), etc.) um diese in Triggervektoren umzuwandeln. Alternativ können von der SYNC-UNIT oder STATE-MACHINE generierte Zustände über 0415 an die BM-UNIT übermittelt werden.

Die von der BM-UNIT an den Bus (0404) übertragenen Triggersignale können je nach Konfiguration der auswertenden konfigurierbaren Elemente dort als STEP/STOP/GO-, RECONFIG-Trigger oder zur Auswahl eines Konfigurationsregisters verwendet werden. Welche Funktion ein generierter Trigger bei den auswertenden konfigurierbaren Elementen erfüllt, wird durch die Vernetzung (0404) und die Konfiguration der jeweiligen konfigurierbaren Elemente bestimmt. Ein und derselbe Trigger kann bei verschiedenen konfigurierbaren Elementen unterschiedliche Funktion haben. 0416 ist der Ergebnisausgang von R-REGsft zum Bussystem 0404 und den nachfolgenden konfigurierbaren Elementen.

Figur 5

In Figur 5 ist das zeitliche Verhalten zwischen generierten Triggern und den durch die Trigger selektierten Konfigurationsregistern beispielsweise aufgezeigt. 0501 generiert durch einen Vergleich den Triggervektor TRIGV, der die Werte "equal" (gleich), "greater" (größer) oder "less" (kleiner) annehmen kann. Die konfigurierbaren Elemente 0502-0504 verarbeiten Daten abhängig vom Vergleich (0501). Dabei ist die Verarbeitung von den Vergleichswerten "equal", "greater" und "less" abhängig. Die Verarbeitung ist gepipelinet, das heißt, ein Datenwort wird nacheinander von 0502, dann von 0503 und zuletzt von 0504 modifiziert.

0505 verarbeitet ebenfalls Daten in Abhängigkeit von 0501. Dabei beschränkt sich die Abhängigkeit allerdings auf die Vergleichswerte "less". "greater" UND "equal" bewirken die gleiche Funktionsausführung. Es werden also die Werte "kleiner" und "größer oder gleich" unterschieden. 0506 ist in der Pipeline 0505 nachgeschaltet. Dabei reagiert 0506 auf "equal", "greater" und "less" unterschiedlich (vgl. 0503). 0507 ist ebenfalls von 0501 abhängig, jedoch werden die Werte "gleich" und "ungleich (kleiner oder größer)" unterschieden. Das Ausführungsbeispiel beginnt bei dem Zeitpunkt t (Fig. 5a) und endet am Zeitpunkt (t+3). Durchlaufen Daten eine der Pipelines (0502, 0503, 0504 bzw. 0505, 0606) werden sie bei jeder Ausführung in einer der Makros (0502-0506) um einen Taktzyklus verzögert. Längere und ins besondere unterschiedliche Verzögerungen können ebenfalls auftreten. Da zwischen den Daten und Triggersignalen ein Handshakemechanismus zur automatischen Synchronisation (gemäß dem Stand der Technik, bzw. dieser Schrift (TRIGACK/TRIGRDY)) besteht, muß auf diesen Fall nicht gesondert eingegangen werden. Durch die Verzögerungen stehen zum Zeitpunkt t beispielsweise zwischen der zweiten und dritten Pipelinestufe die Daten und Triggersignale des früheren Zeitpunktes t-2 an. Von Fig. 5a bis Fig 5d ist der Ablauf von 3 Taktzyklen (t bis t+2) dargestellt.

[0043] Die von 0501 generierten Triggervektoren (also die Vergleichsergebnisse) sehen über t betrachtet wie folgt aus:

Zeit t	Vergleichsergebnis
t-2	less, kleiner
t-1	greater, größer
t	equal, gleich
t+1	greater, größer
t+2	equal, gleich

[0044] In Figur 6 ist die Integration mehrerer Konfigurationsregister-in ein konfigurierbares Element dargestellt. In diesem Ausführungsbeispiel existieren drei Konfigurationsregister (0409) nach Fig. 4. Diese werden über den Bus 0406 konfiguriert. Über das Bussystem 0411 erhält eine Steuereinheit (0601) (die auch als Statemachine ausgestaltet werden kann) die Signale TRIGV und TRIGRDY. Die Steuereinheit schaltet gemäß TRIGV eines der Konfigurationsregister über den Multiplexer (0602) auf das Bussystem 0410, das zu den Steuermechanismen des konfigurierbaren Elementes führt. Zur Synchronisation der Triggersignale mit den internen Abläufen des konfigurierbaren Elementes besitzt 0601 einen Synchronisationsausgang, der an die Synchronisationseinheit (0412) oder die Statemachine (0413) führt. Zur Synchronisation der Triggerquellen generiert 0601 das Handshakesignal TRIGACK nach Verarbeitung des eingegangenen Triggers.

[0045] In dem Ausführungsbeispiel ist jedes der Konfigurationsregister (0409) einem TRIGV vom Typ ("equal", "greater", "less") zugeordnet. Werden bei jedem der Triggertypen andere Operationen ausgeführt, so ist jedes der Konfigu-

rationsregister unterschiedlich belegt. Wird beispielsweise nur aus "equal" und "not equal" unterschieden, sind die Konfigurationsregister für die Typen "less" und "greater" gleich belegt, nämlich mit der Konfiguration für "not equal". Das Konfigurationsregister für "equal" besitzt eine andere Belegung. Das bedeutet, anhand der Belegung der Konfigurationsregister kann der Vergleich genauer spezifiziert werden, wobei jedes konfigurierbare Element diese Spezifizierung unterschiedlich gestalten kann. Über das Register 0603 wird TRIGV zusammen mit dem Ergebnis an die nachfolgenden konfigurierbaren Elemente weitergeleitet um ein Pipelining gem. Fig. 5a-d zu ermöglichen. Das Register und die Handshakesignale werden von 0412 oder.0413 gesteuert. Die Triggerinformation kann zusammen mit dem Ergebnis aus dem R-REGsft oder zeitversetzt, also vor dem Ergebnis, über das Interface 0416 an die nachfolgenden konfigurierbaren Elemente übertragen werden.

Eine zeitversetzte Übertragung bietet den Vorteil, daß keine zusätzliche Zeit zum Einstellen der Konfigurationsregister in den nachfolgenden konfigurierbaren Elementen erforderlich ist, da die Einstellung bereits vor Erhalt der Daten (zeitgleich mit dem Freigeben des Ergebnisses) erfolgt. Ein entsprechendes Timing (bezogen auf bezogen auf DFP übliche Abläufe) ist in Fig. 6a dargestellt. Die Triggervektoren (0615) werden mit der steigenden Flanke (0613) des Bausteintaktes (0614) generiert. Mit der fallenden Flanke (0612) werden die Trigger in den konfigurierbaren Elementen ausgewertet. Die Daten laufen phasenverschoben, d.h. sie werden bei 0612 freigegeben und mit 0613 eingelesen. Während 0610 werden die Triggervektoren über den Bus übertragen und die Daten berechnet. Während 0611 werden die Daten über den Bus übertragen und die Trigger berechnet, bzw. die Konfigurationsregister der konfigurierbaren Elemente gemäß den bei 0613 gespeicherten Daten ausgewählt und die Konfiguration entsprechend eingestellt.

[0046] Figur 7a zeigt die Verwaltung von Sprüngen nach dem Predicate/NOP-Verfahren gemäß dem Stand der Technik. Beim Ausführen eines Vergleiches wird ein Eintrag im Predicate-Register (0704) gesetzt. Dieser Eintrag wird während der Ausführung von Befehlen abgefragt und legt fest, ob ein Befehl ausgeführt wird (der Befehl innerhalb der von dem bedingten Sprung angesprochenen Codesequenz liegt), oder durch einen NOP ersetzt wird (der Befehl liegt in einer anderen als der durch den bedingten Sprung angesprochenen Codesequenz). Der Befehl steht dabei in dem Befehlsregister 0701. Das Predicate-Register enthält eine Mehrzahl von Einträgen, die einer Mehrzahl von Operationen und/oder einer Mehrzahl von Rechenwerken zugeordnet sind. Diese Zuordnung wird zur Compile-Zeit des Programmes vom Compiler vergeben. Die Zuordnungsinformation (0707) wird dem Befehl, der in das Befehlsregister eingetragen wird zugeordnet, so daß ein eindeutiger Eintrag durch den jeweiligen Befehl referenziert wird. Durch 0703 wird ausgewählt ob der Befehl aus 0701 oder ein NOP ausgeführt wird. Bei der Ausführung eines NOPs geht ein Taktzyklus verloren. 0703 hat dabei symbolischen Charakter, da prinzipiell auch die Ausführungseinheit (0702) direkt von 0704 angesteuert werden könnte.

[0047] In Figur 7b existieren n Befehlsregister (0701: Func 1 ... Func n).
Bei der Ausführung eines Vergleiches / bedingten Sprunges wird das zu adressierende Befehlsregister, also das Ergebnis des Vergleiches, als Eintrag (0708) in dem Predicate-Register 0706 abgelegt, wobei 0706 aus einer Mehrzahl solcher Einträge besteht. Der jeweilige Eintrag (0708) in 0706 ist dabei so breit, daß alle möglichen Befehlsregister einer Ausführungseinheit (0702) durch ihn adressiert werden können, das bedeutet, bei Befehlsregistern ist die Eintragsbreite $\log_2(n)$. Das Predicate-Register enthält eine Mehrzahl von Einträgen, die einer Mehrzahl von Operationen und/oder einer Mehrzahl von Rechenwerken zugeordnet sind. Diese Zuordnung wird zur Compile-Zeit des Programmes vom Compiler vergeben. Die Zuordnungsinformation (0707) wird der Menge an Befehlen, die in die Befehlsregister eingetragen wird zugeordnet, so daß ein eindeutiger Eintrag durch die jeweilige Befehle referenziert wird.
Über den Multiplexer wird ausgewählt, welches Befehlsregister den Code für die momentane Ausführung liefert.
Durch diese Technik wird bei bedingten Sprüngen auch im ungünstigsten Fall anstatt eines NOPs ein gültiger Befehl ausgeführt, wodurch kein Taktzyklus verschwendet wird.

[0048] **Vorstehend beschrieben wurde somit unter anderem** ein Verfahren zur Synchronisation sowohl der Ablaufsteuerung der Datenverarbeitung in konfigurierbaren Elementen, als auch deren Umkonfiguration, in Bausteinen mit zwei- oder mehrdimensionaler programmierbarer Zellstruktur (DFP, FPGA, DPGA, RAW-Machines), wobei ein konfigurierbares Bussystem die Elemente miteinander verbindet, als auch der Ablaufsteuerung der Datenverarbeitung in üblichen, auf Rechenwerken aufgebauten Mikroprozessoren, Digitalen Signalprozessoren und Mikrokontrollern, durch bedingte Sprünge, wobei ein Bussystem die Rechenwerke miteinander verbindet, **wobei vorgesehen ist**, daß Synchronisationssignale während der Verarbeitung von den verarbeitenden konfigurierbaren Elementen durch Vergleiche der Daten, Vorzeichen von Zahlen, Überträge von arithmetischen Operationen, Fehlerzuständen, u. dgl. (0202), generiert wer- .

[0049] Operationen, Fehlerzuständen, u. dgl. (0202), generiert werden und an weitere Elemente (0201, 0203) über das Bussystem gesandt werden, wobei die empfangenden Elemente die Information zur Synchronisation der Datenverarbeitung und Steuerung des Ablaufes der Datenverarbeitung verwenden (Fig. 1-3).

[0050] **Weiter ist in diesem Verfahren bevorzugt vorgesehen,** daß bei der Synchronisation durch einen Trigger ein konfigurierbares Element oder Rechenwerk zur Ausführung einer einzigen Operation angeregt werden kann (STEP-Trigger, Fig. 2).

[0051] **Auch ist in diesem Verfahren bevorzugt möglich,** daß bei der Synchronisation durch einen Trigger ein

konfigurierbares Element oder Rechenwerk zur Ausführung einer Vielzahl Operation angeregt werden kann (GO-Trigger, Fig. 2).

[0052] Insbesondere ist auch als möglich vorgesehen, daß bei der Synchronisation durch einen Trigger die Ausführung eines konfigurierbaren Elements oder Rechenwerk angehalten werden kann (STOP-Trigger, Fig. 2).

[0053] Es ist alternativ und/oder zusätzlich möglich, daß bei der Synchronisation durch einen Trigger ein konfigurierbares Element zur Umkonfiguration freigegeben werden kann (Fig. 6).

[0054] Weiter wurde vorstehend unter anderem auch ein Verfahren beschrieben, daß das konfigurierbare Element oder Rechenwerk seinen momentanen Status in einem Statusregister (0402) anzeigt.

[0055] Weiterhin ist hierbei als Möglichkeit vorgesehen, daß die Synchronisationssignale zum Empfänger der Daten, zum Sender der Daten oder zu einem unabhängigen konfigurierbaren Element oder Rechenwerk übertragen werden (Fig. 1-3).

[0056] Auch kann hierbei besonders bevorzugt vorgesehen sein, daß die Übertragung der Synchronisationssignale gesperrt werden kann.

[0057] Insbesondere ist hierbei als bevorzugt möglich, daß unterschiedliche Synchronisationssignale wahlweise übertragen werden (Vergleich, Fehlerzustände, etc.), wobei die Art des Synchronisationssignales frei in der generierenden Einheit wählbar ist und die Auswirkung der Synchronisationssignale frei in der empfangenden Einheit wählbar ist (Fig. 5).

[0058] Weiter wurde hierbei als bevorzugt möglich vorgesehen, daß ein Synchronisationssignal an mehrere Empfänger übertragen werden kann (0501).

[0059] Weiterhin ist hierbei als insbesondere bevorzugt möglich vorgesehen, daß einem Synchronisationssignal eine Quittierungsleitung zugeordnet ist (Fig. 6: TRIGACK).

[0060] Weiter ist bei diesem Verfahren vorgesehen, daß ein Synchronisationsvektor aus einem oder einer Mehrzahl von Synchronisationssignalen aufgebaut ist (Fig. 6: TRIGV).

[0061] Insbesondere ist hierbei als bevorzugt möglich, daß ein Konfigurationsregister aus einer Mehrzahl von Konfigurationsregistern durch einen Synchronisationsvektor ausgewählt wird; bzw. ein Befehlsregister aus einer Mehrzahl von Befehlsregistern ausgewählt wird (Fig. 7b).

[0062] Weiter wurde hier als bevorzugt möglich vorgesehen, daß der Auswahlvorgang eines Registers durch Synchronisationssignale so mit der Datenverarbeitung synchronisiert wird, daß kein Takzyklus verlorengeht (Fig. 6).

[0063] Weiterhin ist hierbei als insbesondere bevorzugt möglich vorgesehen, daß in einem einer Operation zugeordneten Register, ausgewählt aus einer Mehrzahl an Registern, der Wert des generierten Synchronisationssignals so gespeichert wird, daß eine andere zugeordnete Operation selektiv darauf zugreifen kann und anhand der Information einen möglichen und gültigen Befehl, bzw. eine mögliche und gültige Konfiguration, aus einer Mehrzahl von Befehlen/Konfigurationen auswählt.

[0064] Weiter wurde vorstehend unter anderem ein Verfahren zur Synchronisation sowohl der Ablaufsteuerung der Datenverarbeitung in konfigurierbaren Elementen, als auch deren Umkonfiguration, in Bausteinen mit zwei- oder mehrdimensionaler programmierbarer Zellstruktur (DFP, FPGA, DPGA, RAW-Machines), wobei ein konfigurierbares Bussystem die Elemente miteinander verbindet, als auch der Ablaufsteuerung der Datenverarbeitung in üblichen, auf Rechenwerken aufgebauten Mikroprozessoren, Digitalen Signalprozessoren und Mikrokontrollern, durch bedingte Sprünge, wobei ein Bussystem die Rechenwerke miteinander verbindet, **wobei vorgesehen ist,** daß anhand entsprechender Befehle Konfigurationswörter innerhalb eines konfigurierbaren Elementes oder Rechenwerkes generiert werden und über den Datenbus zusammen mit der Adresse des anzusprechenden Registers an ein weiteres konfigurierbares Element oder Rechenwerk übertragen werden, das die übertragenen Konfigurationswörter in das adressierte Register schreibt (Seite 2-5).

[0065] Es ist hierbei weiter als Möglichkeit vorgesehen, daß das konfigurierbare Element oder Rechenwerk seinen momentanen Status in einem Statusregister (0402) anzeigt.

[0066] Weiter wurde hierbei als bevorzugt möglich vorgesehen, daß die Angabe der anzusteuernden Register in Befehlen kodiert ist und über den Datenbus übertragen wird (Seite 4, unten).

[0067] Weiterhin ist hierbei insbesondere bevorzugt möglich, daß in einem einer Operation zugeordneten Register, ausgewählt aus einer Mehrzahl an Registern, der Wert des generierten Synchronisationssignals so gespeichert wird, daß eine andere zugeordnete Operation selektiv darauf zugreifen kann und anhand der Information einen möglichen und gültigen Befehl, bzw. eine mögliche und gültige Konfiguration, aus einer Mehrzahl von Befehlen/Konfigurationen auswählt.

[0068] Weiterhin wurde vorstehend auch beschrieben, ein Verfahren zur Synchronisation sowohl der Ablaufsteuerung der Datenverarbeitung in konfigurierbaren Elementen, als auch deren Umkonfiguration, in Bausteinen mit zwei- oder mehrdimensionaler programmierbarer Zellstruktur (DFP, FPGA, DPGA, RAW-Machines), wobei ein konfigurierbares Bussystem die Elemente miteinander verbindet, sowie der Ablaufsteuerung der Datenverarbeitung in üblichen, auf Rechenwerken aufgebauten Mikroprozessoren, Digitalen Signalprozessoren und Mikrokontrollern, durch bedingte Sprünge, wobei ein Bussystem die Rechenwerke miteinander verbindet, **wobei vorgesehen ist,** daß zur Laufzeit anhand

von Synchronisationssignalen eine gültige Konfiguration der konfigurierbaren Elemente aus einer Mehrzahl an Konfigurationen oder ein gültiger Befehl aus mehreren ein gültiger Befehl aus mehreren möglichen Befehlen eines Rechenwerkes ausgewählt wird (Fig. 7b).

[0069] Insbesondere ist hierbei als bevorzugt möglich, daß das konfigurierbare Element oder Rechenwerk seinen momentanen Status in einem Statusregister (0402) anzeigt.

[0070] Weiter wurde hierbei als bevorzugt möglich vorgesehen, daß die Synchronisationssignale zum Empfänger der Daten, zum Sender der Daten oder zu einem unabhängigen konfigurierbaren Element oder Rechenwerk übertragen werden (Fig. 1-3).

[0071] Auch kann hierbei bevorzugt vorgesehen sein, daß die Übertragung der Synchronisationssignale gesperrt werden kann.

[0072] Weiter ist bei diesem Verfahren bevorzugt vorgesehen, daß unterschiedliche Synchronisationssignale wahlweise übertragen werden (Vergleich, Fehlerzustände, etc.), wobei die Art des Synchronistaionssignales frei in der generierenden Einheit wählbar ist und die Auswirkung der Synchronisationssignale frei in der empfangenden Einheit wählbar ist (Fig. 5).

[0073] Auch ist es bei diesem Verfahren bevorzugt möglich, daß ein Synchronisationssignal an mehrere Empfänger übertragen werden kann (0501).

[0074] Insbesondere ist auch als möglich vorgesehen, daß einem Synchronisationssignal eine Quittierungsleitung zugeordnet ist (Fig. 6, TRIGACK).

[0075] Weiterhin ist hierbei als Möglichkeit vorgesehen, daß ein Synchronisationsvektor aus einem oder einer Mehrzahl von Synchronisationssignalen aufgebaut ist (Fig. 6, TRIGV).

[0076] Insbesondere ist hierbei als bevorzugt möglich, daß der Auswahlvorgang eines Registers durch Synchronisationssignale so mit der Datenverarbeitung synchronisiert wird, daß kein Takzyklus verlorengeht (Fig. 6).

[0077] Weiter wurde hierbei als bevorzugt möglich vorgesehen, daß in einem einer Operation zugeordneten Register, ausgewählt aus einer Mehrzahl an Registern, der Wert des generierten Synchronisationssignals so gespeichert wird, daß eine andere zugeordnete Operation selektiv darauf zugreifen kann und anhand der Information einen möglichen und gültigen Befehl, bzw. eine mögliche und gültige Konfiguration, aus einer Mehrzahl von Befehlen/Konfigurationen auswählt.

Patentansprüche

1. Verfahren zur Synchronisation der Datenverarbeitung zur Laufzeit konfigurierbarer, datenverarbeitender Elemente einer zur Laufzeit rekonfigurierbaren Zellarchitektur mit den Schritten, dass zumindest einige der Elemente konfiguriert werden, ein in einer Zelle erzeugtes Triggersignal zur Laufzeit zu einer Anzahl konfigurierter Elemente innerhalb der Zellarchitektur übertragen wird, das Triggersignal von der Anzahl der Elemente empfangen wird und im Ansprechen auf den Triggerempfang anhand dessen die Ausführung einer Konfiguration zur Datenverarbeitung bestimmt wird.

2. Verfahren nach dem vorhergehenden Anspruch, **dadurch gekennzeichnet, dass** die Konfiguration im Ansprechen auf den Triggereingang die Datenverarbeitung verzögert, sofort und/oder mehrfach ausgeführt wird.

3. Verfahren nach einem der vorhergehenden Ansprüche, **dadurch gekennzeichnet, dass** zumindest einige der Elemente durch Einschreiben einer Mehrzahl von Konfigurationen in Konfigurationsregister vorkonfiguriert werden, ein in einer Zelle erzeugtes Triggersignal zu einer Anzahl vorkonfigurierter Elemente innerhalb der Zellarchitektur zur Laufzeit übertragen wird, das Triggersignal von der Anzahl Elemente empfangen wird und anhand dessen eine gültige Konfiguration aus der Mehrzahl von in den Konfigurationsregistern vorab abgelegten Konfigurationen ausgewählt wird und Daten von zumindest einem vorkonfigurierten Element der Anzahl gemäß seiner aus der Vorkonfiguration ausgewählten gültigen Konfiguration verarbeitet werden.

4. Verfahren zur Synchronisation nach Anspruch 1 mit dem Schritt, dass ein Triggervektor erzeugt wird und ein triggervektorbezogenes Signal übertragen wird, um die Datenverarbeitung zu synchronisieren.

5. Verfahren nach einem der vorhergehenden Ansprüche, **dadurch gekennzeichnet, dass** das Triggersignal gemäß einer vorkonfigurierten Vernetzung übertragen wird.

6. Verfahren nach einem der vorhergehenden Ansprüche, **dadurch gekennzeichnet, dass** sequenziell eine Rekonfiguration von Elementen der Anzahl als eine Funktion des empfangenen Triggers ausgelöst wird.

7. Verfahren nach einem der vorhergehenden Ansprüche, **dadurch gekennzeichnet, dass** Daten entlang eines vorkonfigurierten Weges propagiert werden, ein Triggersignal entlang eines vorkonfigurierten Weges propagiert wird und eine Rekonfiguration ausgelöst wird als Funktion der Triggerpropagation.

8. Verfahren nach einem der vorhergehenden Ansprüche, worin die Datenverarbeitung im Ansprechen auf die Abwesenheit eines Triggersignals erfolgt.

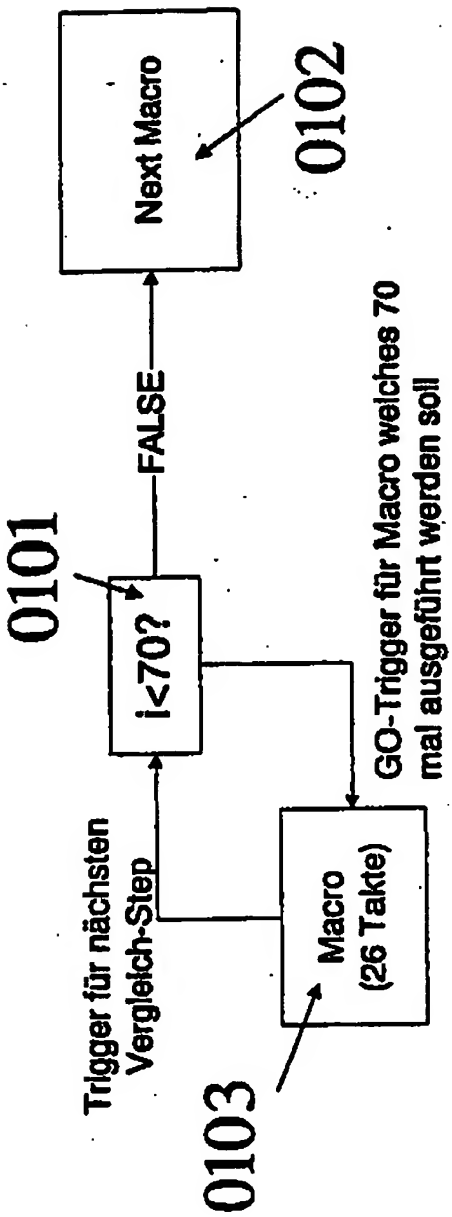

Next Macro
0102
FALSE
0101
i<70?
GO-Trigger für Macro welches 70 mal ausgeführt werden soll
Trigger für nächsten Vergleich-Step
Macro (26 Takte)
0103

Fig. 1

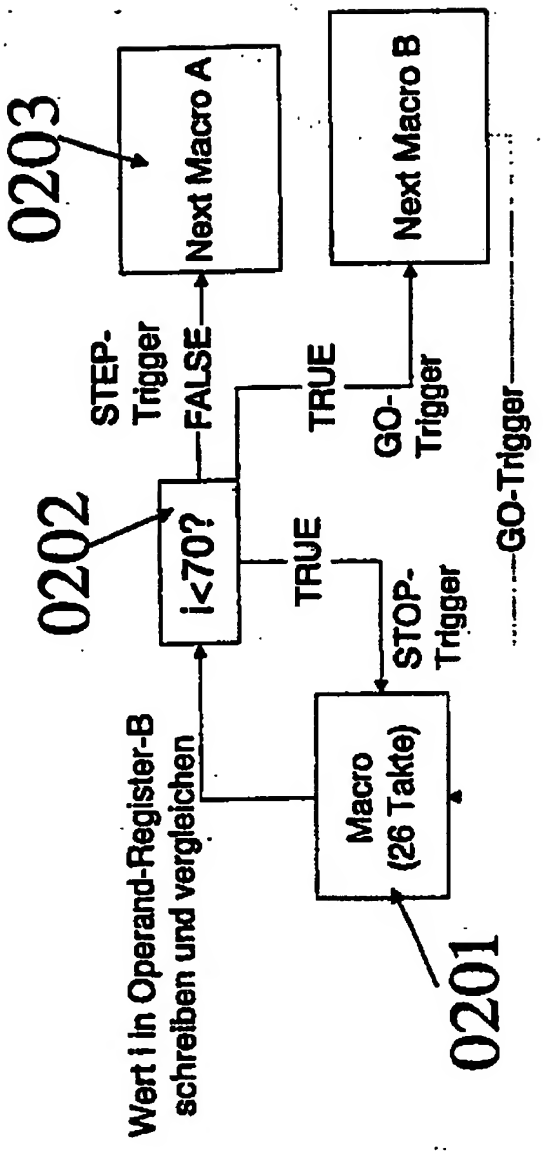

0203
Next Macro A
Next Macro B
STEP-Trigger
FALSE
TRUE
GO-Trigger
0202
i<70?
TRUE
STOP-Trigger
GO-Trigger
Wert i in Operand-Register-B schreiben und vergleichen
Macro (26 Takte)
0201

Fig. 2

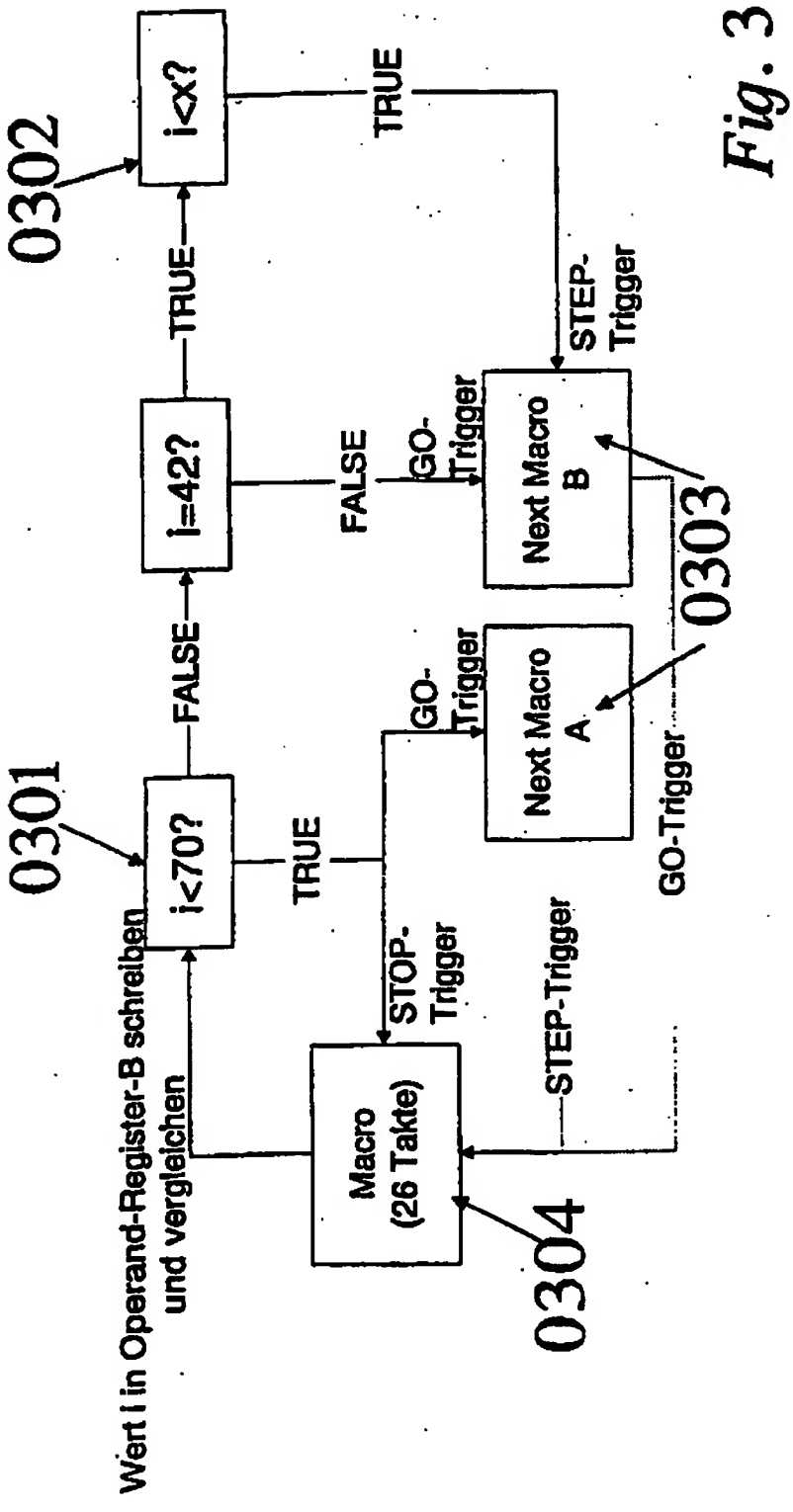

0302
i<x?
TRUE
TRUE
i=42?
FALSE
GO-Trigger
Next Macro B
STEP-Trigger
0303
FALSE
0301
i<70?
TRUE
GO-Trigger
Next Macro A
GO-Trigger
Wert I in Operand-Register-B schreiben und vergleichen
STOP-Trigger
STEP-Trigger
Macro (26 Takte)
0304

Fig. 3

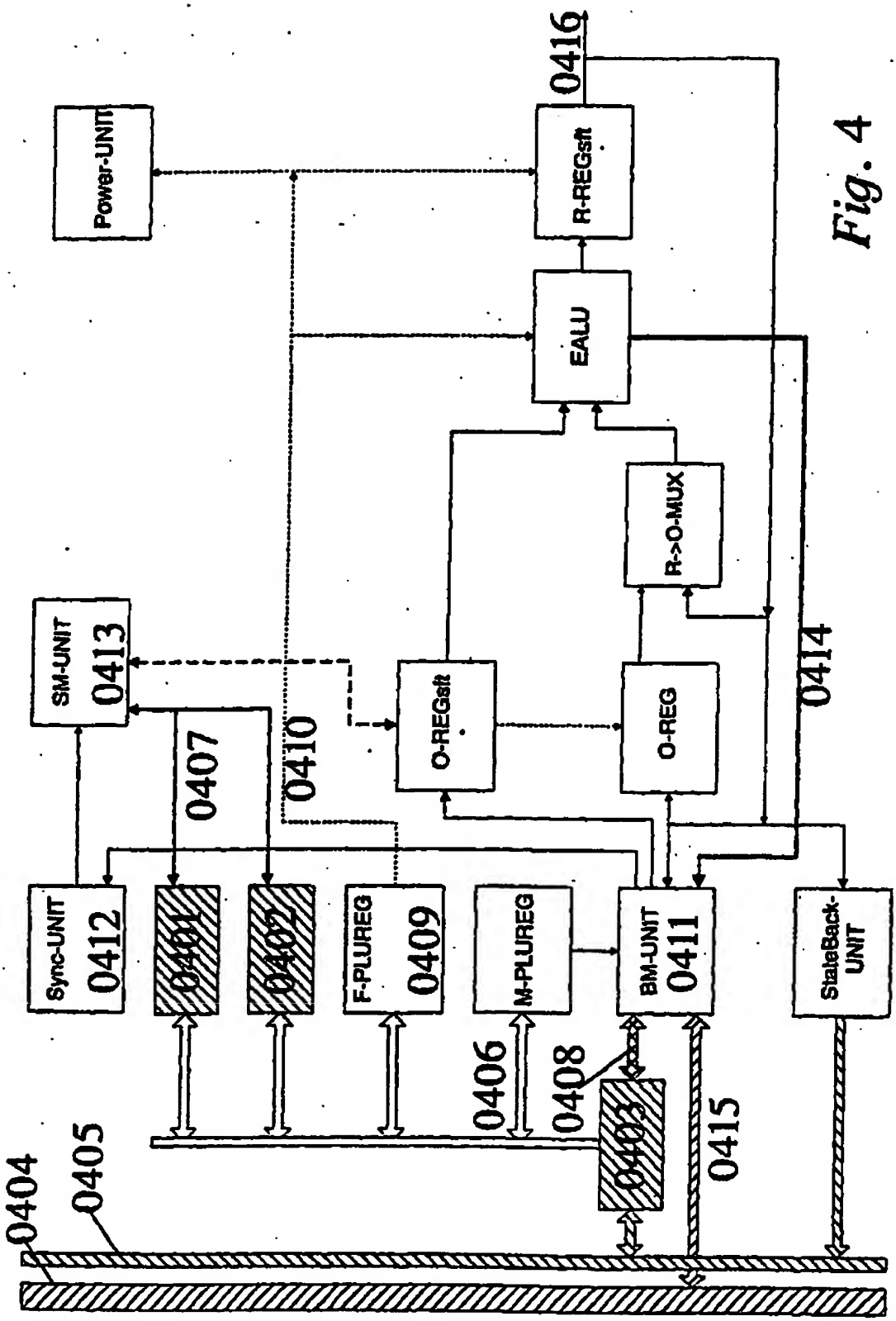

Power-UNIT
0416
R-REGsft
EALU
R->O-MUX
SM-UNIT
0413
0407
0410
O-REGsft
O-REG
0414
Sync-UNIT
0412
0401
0402
F-PLUREG
0409
M-PLUREG
BM-UNIT
0411
StateBack-UNIT
0406
0408
0403
0415
0404
0405

Fig. 4

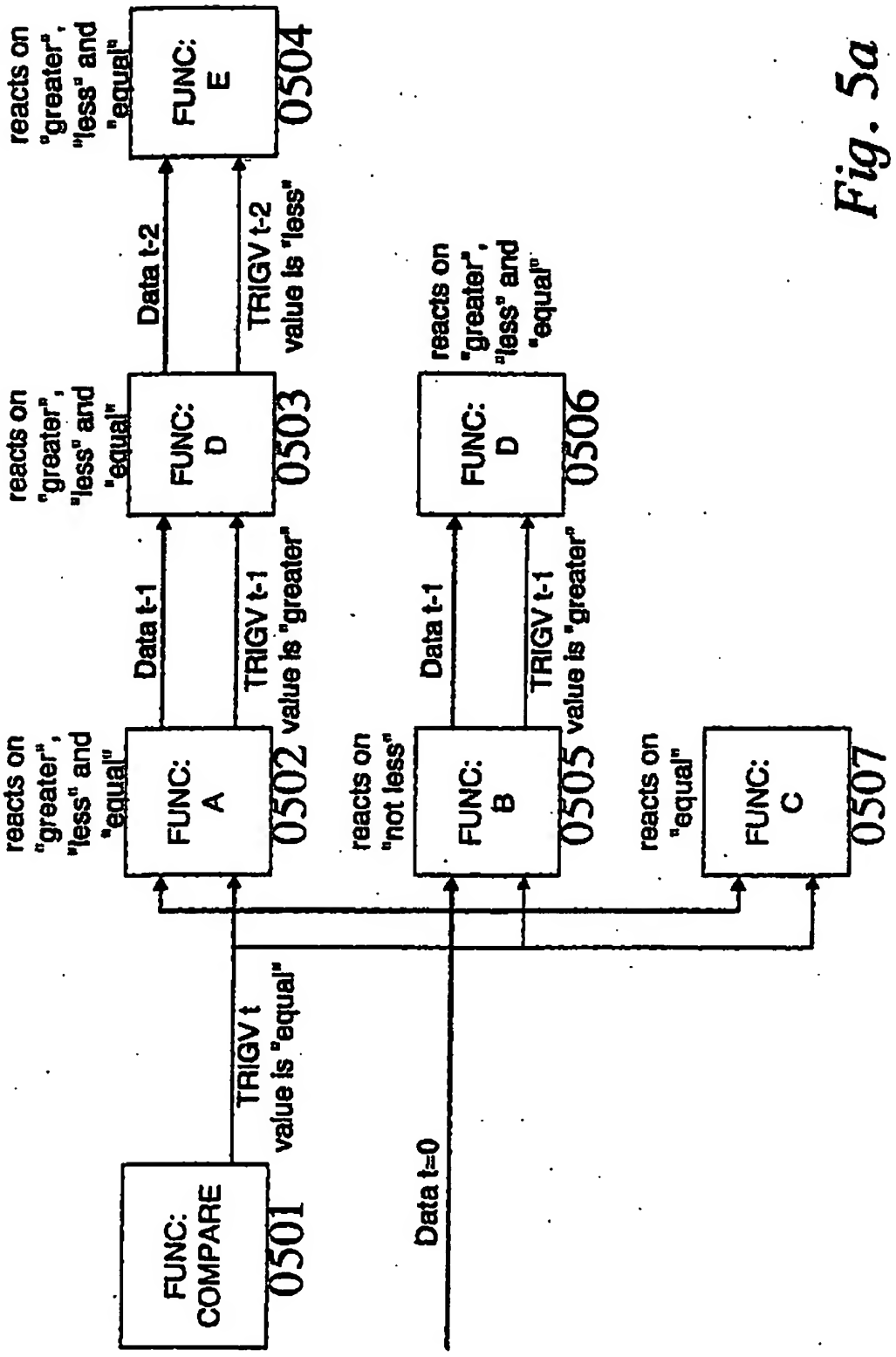

FUNC: COMPARE
0501
TRIGV t value is "equal"
Data t=0
FUNC: A
0502
reacts on "greater", "less" and "equal"
Data t-1
TRIGV t-1 value is "greater"
FUNC: B
0505
reacts on "not less"
FUNC: C
0507
reacts on "equal"
FUNC: D
0503
reacts on "greater", "less" and "equal"
Data t-2
TRIGV t-2 value is "less"
FUNC: E
0504
reacts on "greater", "less" and "equal"
FUNC: D
0506
reacts on "greater", "less" and "equal"

Fig. 5a

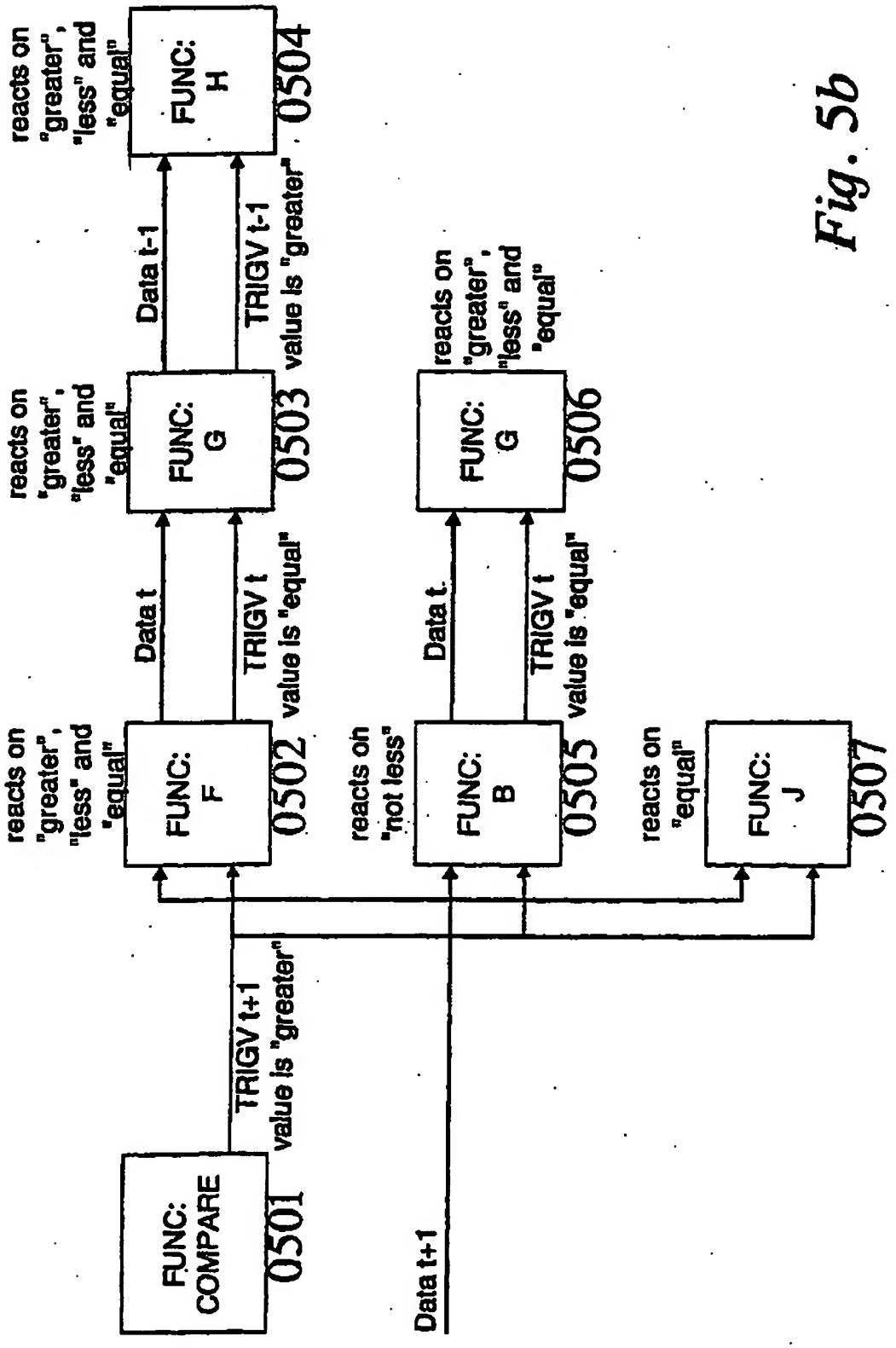

FUNC: COMPARE
0501
TRIGV t+1 value is "greater"
Data t+1
reacts on "greater", "less" and "equal"
FUNC: F
0502
Data t
TRIGV t value is "equal"
reacts on "not less"
FUNC: B
0505
reacts on "equal"
FUNC: J
0507
reacts on "greater", "less" and "equal"
FUNC: G
0503
Data t-1
TRIGV t-1 value is "greater"
reacts on "greater", "less" and "equal"
FUNC: H
0504
reacts on "greater", "less" and "equal"
FUNC: G
0506

Fig. 5b

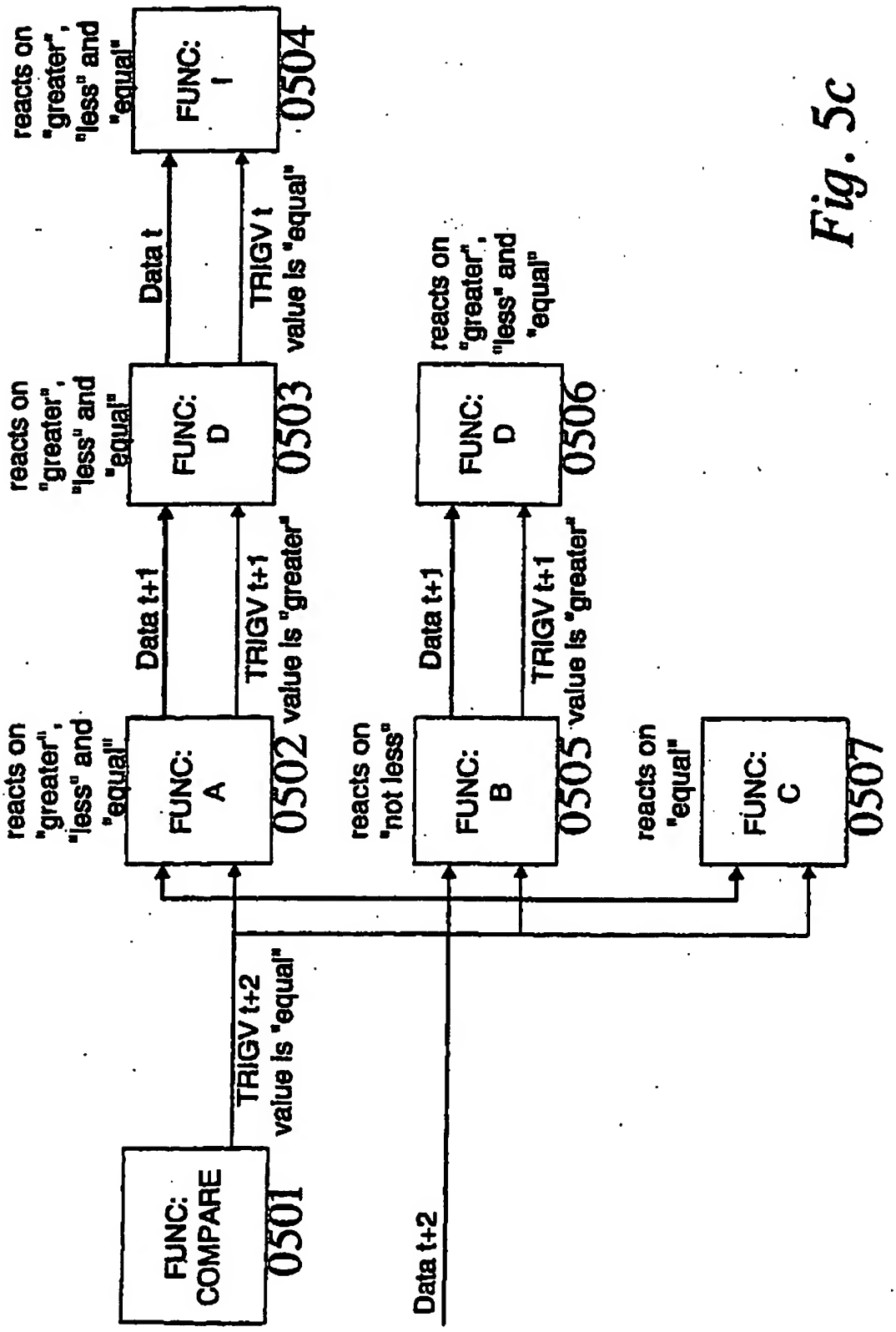

FUNC: COMPARE
0501
TRIGV t+2 value is "equal"
Data t+2
reacts on "greater", "less" and "equal"
FUNC: A
0502
Data t+1
TRIGV t+1 value is "greater"
reacts on "not less"
FUNC: B
0505
Data t+1
TRIGV t+1 value is "greater"
reacts on "equal"
FUNC: C
0507
reacts on "greater", "less" and "equal"
FUNC: D
0503
Data t
TRIGV t value is "equal"
reacts on "greater", "less" and "equal"
FUNC: D
0506
reacts on "greater", "less" and "equal"
FUNC: I
0504

Fig. 5c

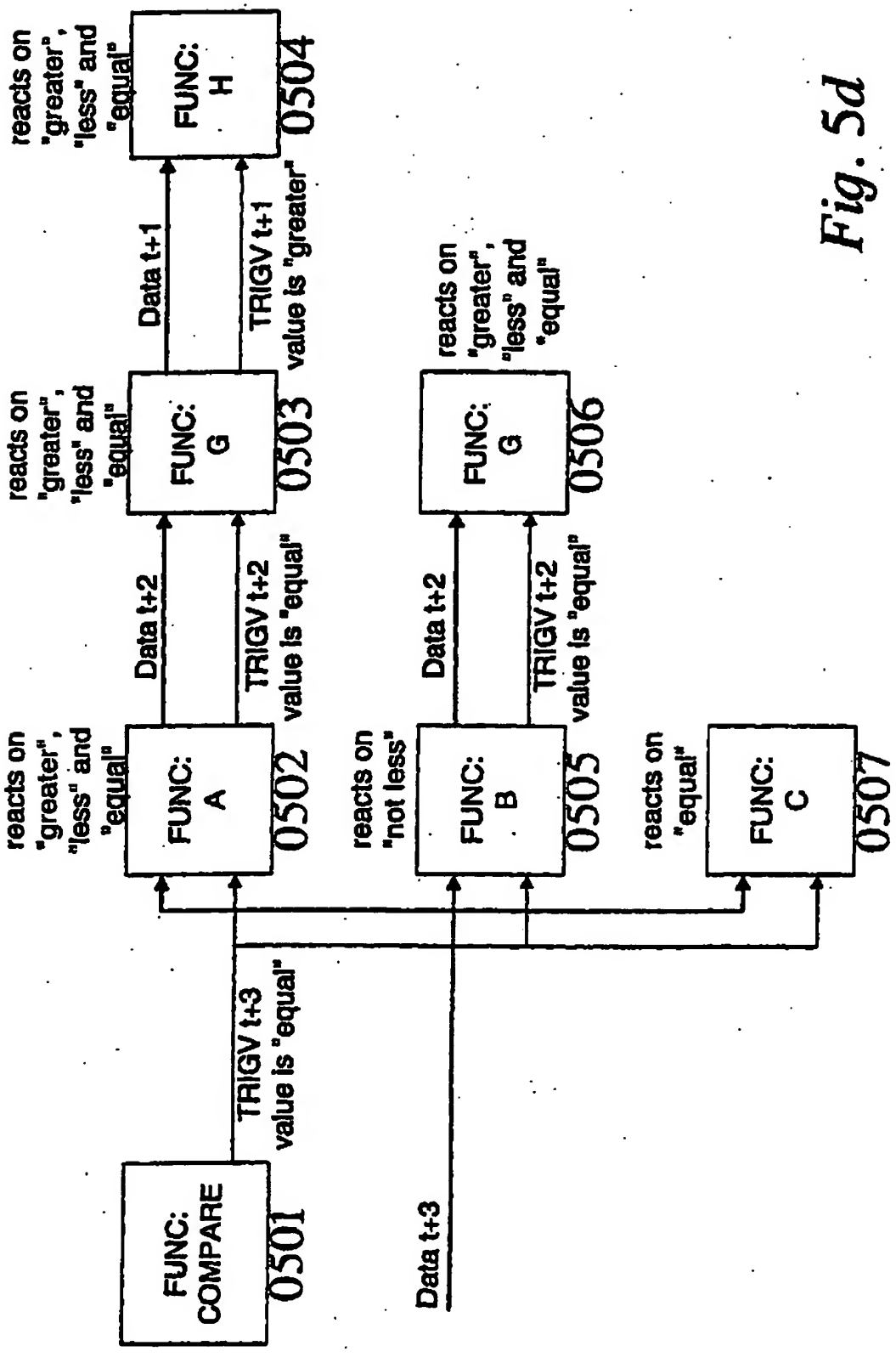

FUNC: COMPARE
0501
TRIGV t+3 value is "equal"
Data t+3
reacts on "greater", "less" and "equal"
FUNC: A
0502
Data t+2
TRIGV t+2 value is "equal"
reacts on "not less"
FUNC: B
0505
Data t+2
TRIGV t+2 value is "equal"
reacts on "equal"
FUNC: C
0507
reacts on "greater", "less" and "equal"
FUNC: G
0503
Data t+1
TRIGV t+1 value is "greater"
reacts on "greater", "less" and "equal"
FUNC: G
0506
reacts on "greater", "less" and "equal"
FUNC: H
0504

Fig. 5d

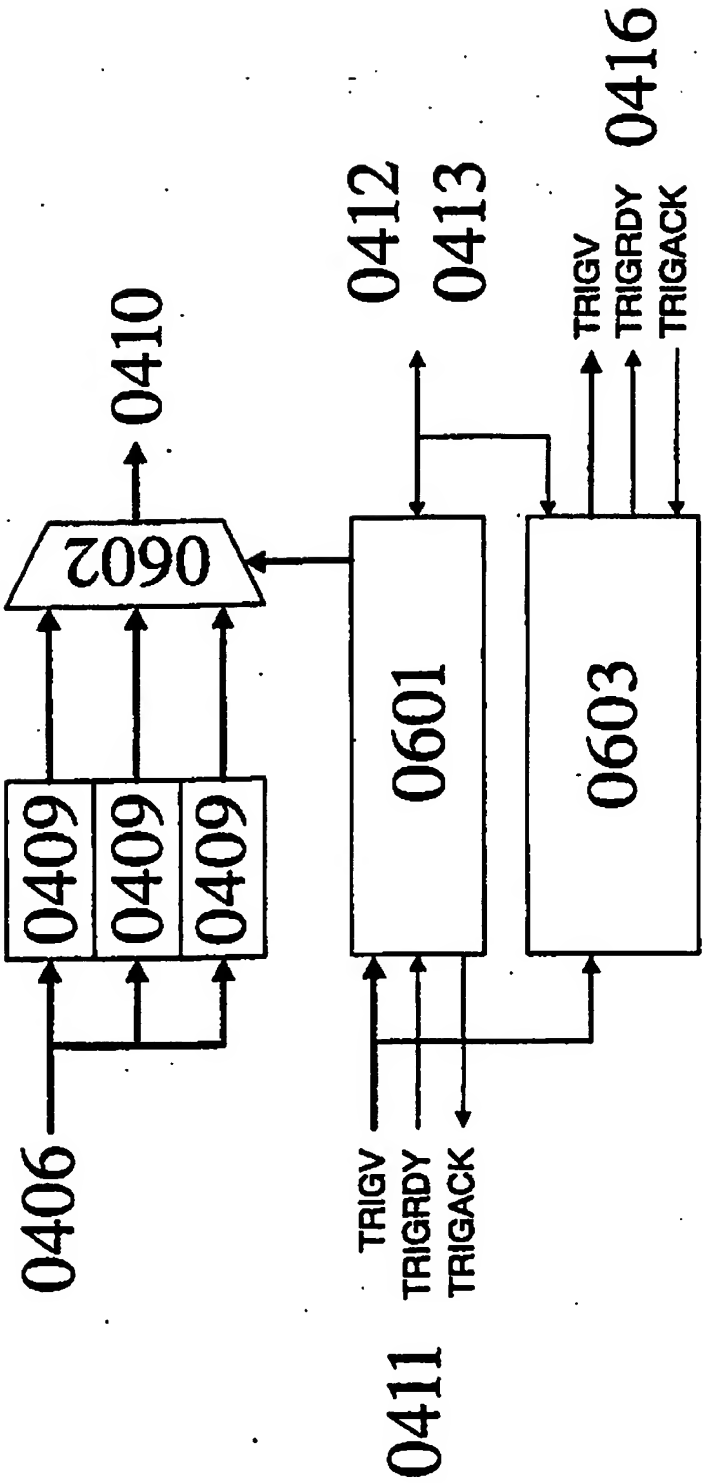

0406
0409
0409
0409
0602
0410
0601
0603
0411
TRIGV
TRIGRDY
TRIGACK
0412
0413
TRIGV
TRIGRDY
TRIGACK
0416

Fig. 6

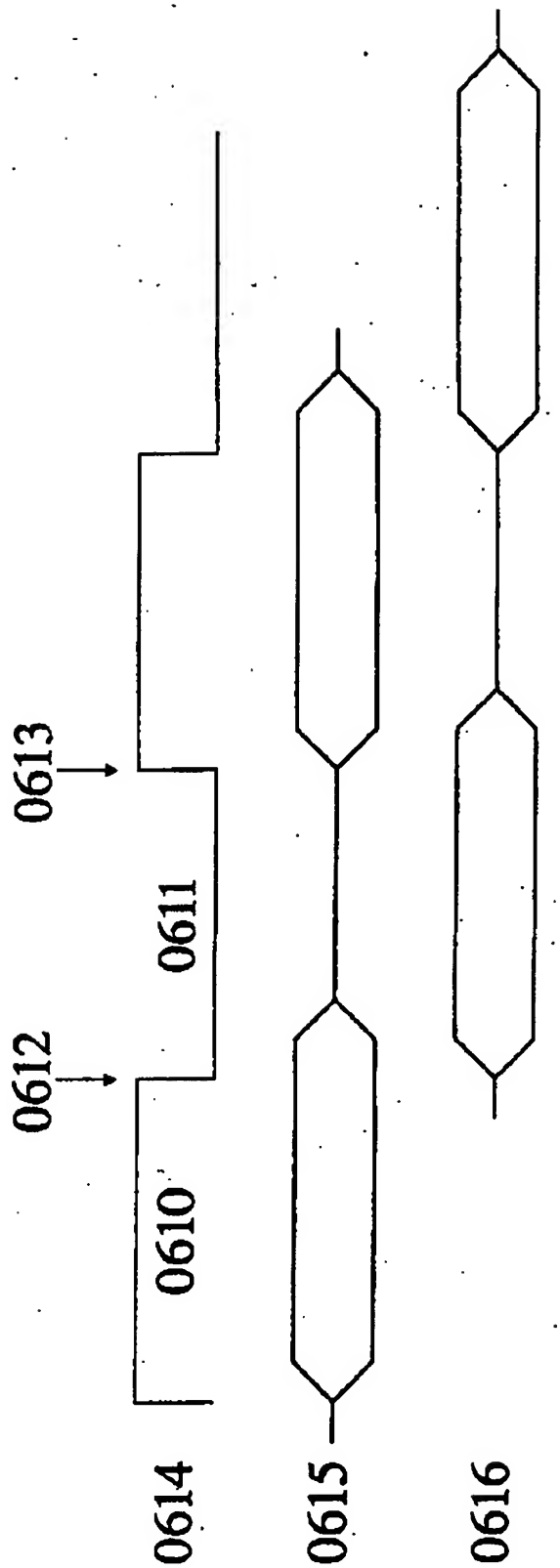


Fig. 6a

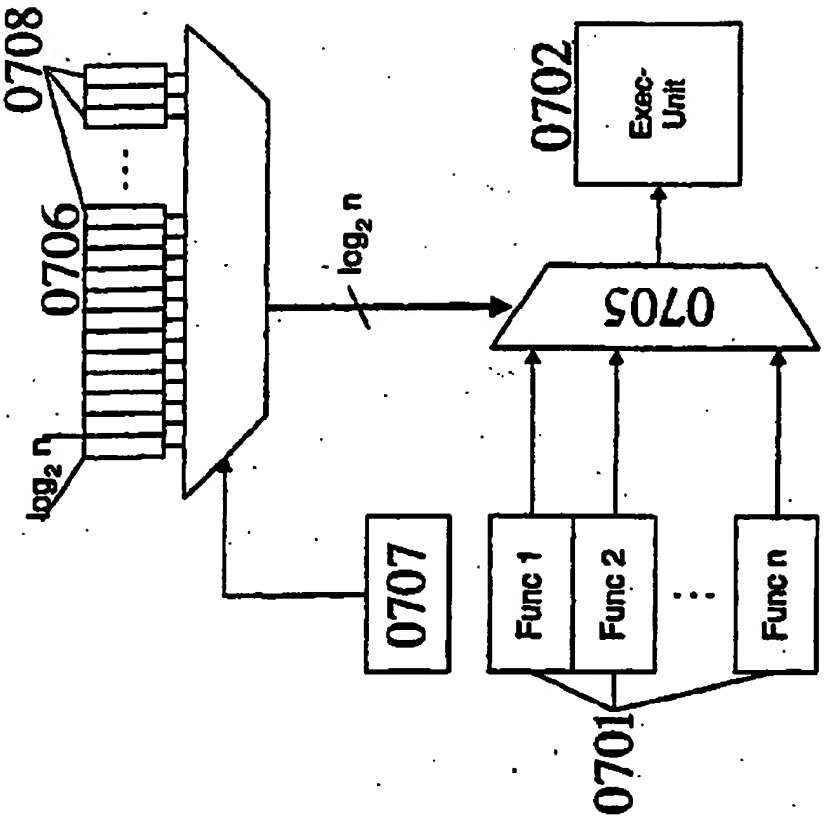


Fig. 7b

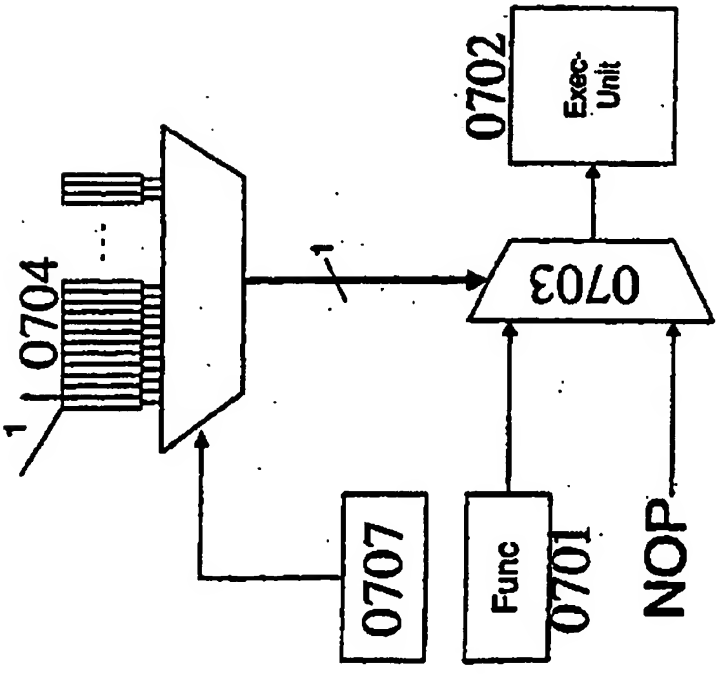
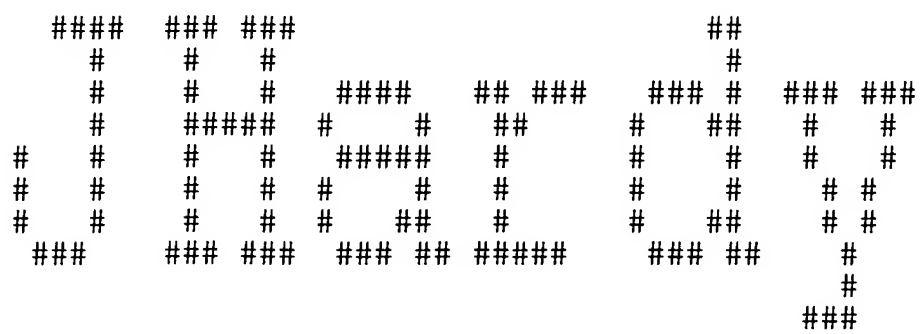
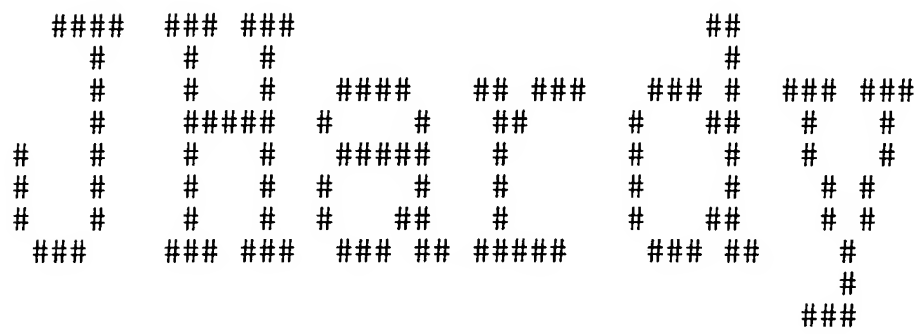


Fig. 7a
prior art

****
****





****
****

Print Job Information:

Date: 1/8/2009

Time: 7:43:52 PM

Job Number: 633

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : H03K 19/177

A1

(11) International Publication Number: **WO 98/10517**

(43) International Publication Date: 12 March 1998 (12.03.98)

(21) International Application Number: PCT/US97/10279

(22) International Filing Date: 16 June 1997 (16.06.97)

(30) Priority Data:
08/708,247 3 September 1996 (03.09.96) US

(71) Applicant: XILINX, INC. [US/US]; 2100 Logic Drive, San Jose, CA 95124 (US).

(72) Inventor: YOUNG, Steven, P.; 6131 Paso Los Cerritos, San Jose, CA 95120 (US).

(74) Agents: YOUNG, Edel, M. et al.; Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124 (US).

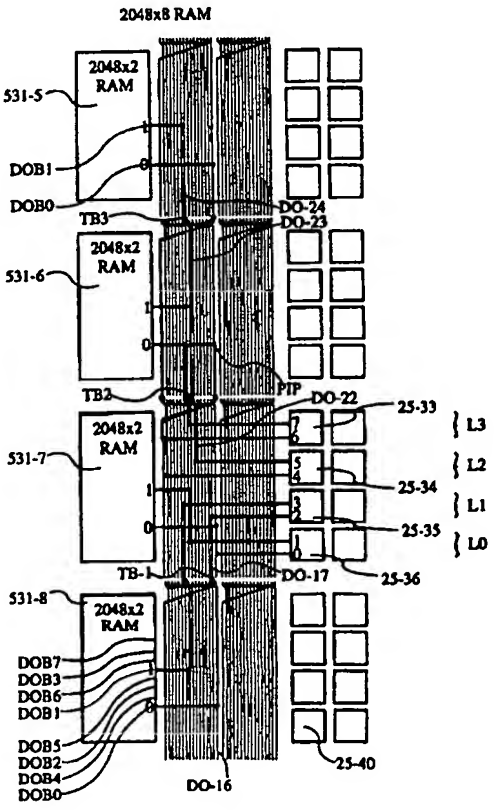
(81) Designated States: JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
With amended claims.

(54) Title: FPGA ARCHITECTURE HAVING RAM BLOCKS WITH PROGRAMMABLE WORD LENGTH AND WIDTH AND DEDICATED ADDRESS AND DATA LINES

(57) Abstract

A structure in which blocks of random access memory, or RAM, are integrated with FPGA configurable logic blocks. Routing lines which access configurable logic blocks also access address, data, and control lines in the RAM blocks. Thus, the logic blocks of the FPGA can use these routing lines to access portions of RAM. In one embodiment, dedicated address and data lines access the RAM blocks of the present invention and are connectable to routing lines in the interconnect structure. These lines allow RAM blocks and arrays of RAM blocks to be configured long, wide, or in between, and allow logic blocks to conveniently access RAM blocks in a remote part of the chip. Access to the RAM blocks is efficient in any RAM configuration. Bidirectional buffers or pass devices segment the address and data lines at each RAM block so that a selectable number of RAM blocks can operate together as a RAM. In another embodiment, dedicated data lines are programmably connectable in a staggered arrangement so that RAM blocks can be connected over a long distance without conflict between the RAM blocks.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

FPGA ARCHITECTURE HAVING RAM BLOCKS WITH PROGRAMMABLE WORD LENGTH AND WIDTH AND DEDICATED ADDRESS AND DATA LINES

RELATED PATENT APPLICATIONS

The present application relates to the following patent applications, all assigned to Xilinx, Inc, assignee of the above application, which are incorporated herein by reference:

1. Pending U.S. application serial no. 08/222,138 [M-2257-1N] entitled "Tile Based Architecture for FPGA" which was filed by Danesh Tavana, Wilson K. Yee, and Victor A. Holen on April 1, 1994 and which is incorporated herein by reference.

2. U.S. Patent 5,517,135 [X-195] entitled "Bidirectional Tristate Buffer with Default Input" which was filed 07-26-95 by Steven P. Young and issued 05-14-96.

FIELD OF THE INVENTION

The invention relates to programmable logic devices formed in integrated circuits and more particularly to field programmable logic devices or field programmable gate arrays.

BACKGROUND OF THE INVENTION

A field programmable gate array (FPGA) comprises an array of programmable logic blocks which can be programmably interconnected to each other to provide a logic function desired by a user. U. S. Patent 4,870,302, reissued as U. S. Patent Re34,363 to Ross Freeman describes the first FPGA, and is incorporated herein by reference. Later patents such as U. S. Patents 4,758,745 to Elgamal, and 5,243,238 to Kean and published application WO 93/05577 invented by Furtek and owned by Concurrent Logic, Inc. describe other FPGA architectures. These patents and application are also incorporated herein by reference. The Xilinx 1994 Data Book entitled "The

Programmable Logic Data Book", available from Xilinx, Inc., 2100 Logic Drive, San Jose, California 95124 describes several FPGA products. As illustrated in the Xilinx data book, for example at page 2-114, FPGA products typically include a regular array of logic blocks, the number of which varies from one product to another.

An FPGA architecture includes a programmable routing structure and an array of configurable logic blocks. The programmable routing matrix includes means for connecting logic blocks to each other. Thus an FPGA provides a combination of programmable logic and programmable connections to a general routing structure.

In a typical FPGA application, the PIPs are turned on ahead of time by loading appropriate values into configuration memory cells associated with the PIPs, thus creating paths and establishing the logic performed by the configurable logic blocks. During operation, signals on the paths change dynamically as values are being written to and read from flip flops.

Some users need blocks of random access memory (RAM), for example so that complex functions generated in one part of the FPGA chip can be synchronized with complex logic generated in another part of the chip. For another example, users may want to provide a FIFO (first-in-first-out register) for buffering high speed data onto and off the chip, or to provide register banks for use by other logic in the chip.

In conventional FPGA chips these blocks of RAM are generated by configuring programmable parts of the FPGA, thus making these parts of the FPGA unavailable for other uses. When a common function such as RAM is desired by many users, it becomes economical to dedicate a portion of the chip to this purpose, thus allowing the particular function to be implemented at high density and leaving other parts of the FPGA free for less predictable uses.

The Altera FLEX 10K chip includes blocks of RAM that can be accessed by logic blocks in the chip. The Altera FLEX 10K structure is described briefly in a product

information bulletin from Altera Corp. dated January 1996 and entitled "Benefits of Embedded RAM in FLEX 10K Devices". A block diagram on this publication shows a RAM/ROM block with several configurations. The RAM/ROM block is in an EAB (embedded array block) that includes input flip flops, a write pulse circuit, and input multiplexers for generating data, address, and write-enable signals from data, address, write enable, and input clock signals. The EAB also includes data-out flip flops and multiplexers for generating data-out signals. The Altera publication indicates that when large RAMs are desired, EABs are cascaded to implement larger RAMs.

However, these dedicated RAM blocks are accessed through general interconnect lines, and using general interconnect lines to access RAM decreases the availability of general interconnect lines for routing other logic signals.

SUMMARY OF THE INVENTION

According to the present invention, blocks of random access memory, or RAM, integrated with FPGA configurable logic blocks include dedicated routing lines. General routing lines which access configurable logic blocks also access address, data, and control lines in the RAM blocks through the dedicated routing lines. Thus, the logic blocks of the FPGA can use these general and dedicated routing lines to access portions of RAM. These dedicated routing lines allow efficient connection of RAM blocks and arrays of RAM blocks whether the RAM blocks are configured long, wide, small, or large. Also, the dedicated routing lines allow logic blocks to conveniently access RAM blocks in a remote part of the chip. Access to the RAM blocks is efficient in any RAM configuration. Bidirectional buffers or pass devices segment the address and data lines at each RAM block so that a selectable number of RAM blocks can operate together as a RAM.

In one embodiment of the invention, adjacent RAM blocks are joined through tristate bidirectional buffers.

These buffers include a default input such that the signal applied to one of the lines connected to the bidirectional buffer is always applied to the input terminal of a buffer element and applied by the output terminal of the buffer element to any load which may be connected to the buffer output terminal. Thus the buffer is never left floating, and the load connected to the output terminal of the buffer switches more quickly than if a signal had to flow through another pass transistor as with a symmetrical bidirectional buffer. In a preferred embodiment, the tristate bidirectional buffer with default input requires only four routing transistors plus the transistors which comprise the memory cells and the buffer element. Further, attaching a load to the output terminal of the buffer element may allow the direction control transistors to be smaller for the same switching speed than attaching the load directly to the line carrying the signal to be buffered. This buffer is further described in U.S. Patent 5,517,135 [X-195].

According to yet another aspect of the invention, a decoder which enables dedicated RAM is configurable to respond in many different ways to decoder input signals. The decoder can be programmed to be enabled by any combination of decoder input signals and can be programmed to ignore any number of decoder input signals. The ability to ignore input signals is important in FPGAs because it saves having to route a disabling signal to an unused decoder input terminal. The decoder can also be programmed to be disabled regardless of decoder input signals. It can be programmed to treat a set of input signals as an address. It can invert or not invert the address.

In one embodiment, data bus line segments in one RAM block are joined to line segments in another RAM block in a staggered or rotating fashion. For example, line 8 in one RAM block is joined to line 7 in the next RAM block. This means that several RAM blocks can be configured to be deep but not wide, so the RAM in one RAM block does not

use all its own data lines and adjacent RAM blocks can use some of the lines, and data in these adjacent RAM blocks can be conveniently accessed on data lines which do not contend.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1A shows an FPGA chip which includes RAM blocks according to the present invention.

Figs. 1B and 1C illustrate two embodiments of RAM block 13 of Fig. 1A, showing external connections only.

Fig. 2 shows the internal connections of the RAM block of Fig. 1B, including a RAM.

Figs. 3A-3D illustrate some of the symbols used in Figs. 2 and 7A.

Figs. 4A and 4B illustrate bidirectional buffer symbols used in Figs. 1B, 1C and 2, and a related bidirectional buffer usable with the present invention.

Fig. 5 shows one embodiment of a decoder according to the present invention which may be used in the RAM block of Fig. 2.

Figs. 6A-6C show four adjacent RAM blocks of the embodiment in Fig. 2, configured in three possible configurations to form RAMs of different word lengths.

Fig. 7A shows the internal connections of the RAM block of Fig. 1C.

Figs. 7B and 7C show enlargements of portions of Fig. 7A near RAM 531.

Figs. 8A-8C represent RAMs configured from four adjacent RAM blocks of the embodiment in Figs. 7A-7C.

DETAILED DESCRIPTION

Fig. 1A shows an FPGA chip according to the present invention. In the center portion of the chip are a plurality of logic blocks 1, separated periodically by columns of RAM blocks 13. Each RAM block spans the height of four logic blocks 1. (In another embodiment, a RAM block can span fewer or more logic blocks, and columns of RAM blocks need not extend the full height of the FPGA

chip.) Along the four edges of the chip are edge blocks 2 which include input/output blocks. Also present on the chip are pads (not shown) for connecting to external pins of a package which holds the chip. The pads connect to edge blocks 2, and each edge block is connected to at least one logic block 1. Also present are RAM edge blocks 4 which connect to RAM blocks 13 and corner blocks 5 which connect to edge blocks 2 and which typically provide certain global signals such as a global clock signal and boundary scan signals. Edge blocks 2, corner blocks 5, and logic blocks 1 are discussed in detail in application serial no. 08/222,138 [M-2257-1N], incorporated herein by reference, and are not again described here.

Group 101 of RAM blocks will be discussed later in connection with Figs. 6A-6C.

Fig. 1B shows a first embodiment of a RAM block 13 of Fig. 1A, and shows external signal lines or buses connected to RAM block 13. This RAM block includes 2304 memory cells configured as a 256x9 RAM. RAM block 13 is the height of four logic blocks 1. Typically, a logic block 1 must access a RAM block 13 either above or below its own position. Extending into RAM block 13 are four sets of routing lines L0 through L3 from adjacent logic blocks, one set of lines associated with each row of adjacent logic blocks. To facilitate access to RAM blocks above or below these horizontal lines, dedicated vertical lines carry address and data signals to the RAM blocks. Address bus ADDR, data-in bus DIN, and data-out bus DOUT extend vertically through the RAM block and programmably connect to the next adjacent RAM block. Also, extending vertically into RAM block 13 are global and local clock lines GCLK and CLK, write enable line WE, and block enable line EN. Global clock line GCLK is permanently connected to RAM blocks throughout the chip and other lines are programmably connectable to the RAM block above.

Fig. 1C shows a second embodiment of a RAM block 13 of Fig. 1A. In Fig. 1C, no enable line EN is provided. Instead the routing of the data-in and data-out lines

makes an enable line unnecessary. Instead of a write enable line, the embodiment of Fig. 1C includes a write enable bus 33, in one embodiment 4 lines wide. In both Fig. 1B and Fig. 1C, separate data-in and data-out buses are provided.

In the Fig. 1B embodiment, the data-out buses of several RAM blocks can be connected together, in which case the same lines receive data out signals from several RAM blocks. Contention is avoided by activating the enable signal for only one commonly connected RAM block at one time. In Fig. 1C, several RAM blocks can be connected together and contention is avoided by staggering or rotating the data line connections so that signals to or from the several RAM blocks do not appear on the same line. The RAM block of Fig. 1C includes 4096 memory cells which can have four different internal configurations 4096x1, 2048x2, 1024x4, and 512x8, whereas the RAM block of Fig. 1B has one internal configuration 256x9. Other configurations are possible. For efficiency, the width options in Fig. 3 should be powers of 2 whereas in Fig. 1B, any width may be used. Detail of Fig. 1B is discussed below in connection with Fig. 2. Detail of Fig. 1C is discussed below in connection with Figs. 7A-7C.

First Embodiment, Fig. 2

Fig. 2 shows in detail dual port RAM block 13 of Fig. 1B. This RAM block includes a dual port RAM 131. A dual port RAM provides two independent sets of address, data, and control lines for accessing the same memory locations. A dual port RAM is useful when a user wants to read or write to two addresses in the same memory at the same time. For example, a FIFO which receives bursts of data from an external source and stores the data for internal processing makes good use of the dual ports, using one port for being written to from the external source and the other for being read by internal logic.

In Fig. 2, as in Fig. 1B, extending horizontally in dual port RAM block 13 are four sets of local interconnect

lines L0 through L3 which connect to adjacent logic blocks. Extending vertically in dual port RAM block 13 are the address, data, and control lines of the RAM block. Four global clock lines GCLK connect permanently to RAM blocks above and below the illustrated RAM block 13. Two local clock lines LCLK connect programmably to RAM blocks above and below. Both clock lines GCLK and LCLK connect programmably to RAM 131. The programmable connections shown at the top edge of Fig. 1B are represented by pentagon symbols to the left of RAM 131. The pentagon symbol is discussed below in connection with Fig. 4A. Write enable lines WEA and WEB and address enable buses ENB and ENA connect programmably to RAM blocks above and below as do address buses ADDRB and ADDRA.

Many configurations of the FPGA are implemented more conveniently with separate data-in and data-out lines because an interconnect line must be driven from a single source, and if a RAM must have tristate control on all data output drivers, it may be difficult to route a signal for providing the tristate control to the RAM. Thus it is preferable for data-in and data-out signals not to share the same line. Therefore, dual port RAM block 13 includes both data-in buses DINB and DINA for writing to RAM 131 and data out buses DOUTB and DOUTA for reading from RAM 131. These data buses are connectable to RAMs above and below the one shown. In one embodiment the LCLK, WEA, WEB, ENA. and ENB lines and the ADDRA, ADDRB, DINA and DINB buses use buffered connections (represented by pentagons and discussed below), and the DOUTA and DOUTB buses use unbuffered connections (represented by open circles).

Fig. 3A illustrates that the circle surrounding an intersection of two lines represents a pass transistor controlled by a memory cell M for connecting the two lines. Fig. 3B illustrates that the open circle joining two line segments also represents a pass transistor controlled by a memory cell for connecting the two line segments.

Fig. 3C illustrates that the triangle pointing in the direction of one line segment represents a programmable connection which applies a signal from the line at the flat edge of the triangle onto the line pointed to by the triangle. (The signal is of course then present on the full length of the line. A triangle pointing in the opposite direction would have the same meaning since it would point to the same wire.) Where signal flow is always in the same direction, the triangle symbol assists the reader with following the flow of signals. A simple pass transistor can make the connection, as shown in Fig. 3C.

When one line is accessed by several programmable connections, these are frequently implemented as a multiplexer. Fig. 3D shows four vertical lines V1 through V4 that can each place a signal onto horizontal line H. Only one of these programmable connections is used at one time. As shown at the right of Fig. 3D, two memory cells M1 and M2 select which of vertical lines V1 through V4 is connected through buffer B to horizontal line H.

Bidirectional Buffer

Fig. 4A illustrates that the pentagon between line segments A and B which is partly covered by line segment A is a bidirectional buffer, and illustrates another aspect of the buffer in which the bidirectional buffered connection between line segments includes a default input from line A to the buffer when the buffer is in a tristate mode.

As shown in Fig. 4A, the bidirectional buffer includes one buffer element 41. Two memory cells M1 and M2 control the direction of signal flow and whether the buffer is in a tristate mode. When M1 and M2 both hold logical 0, the buffer is in a tristate mode, driving neither of line segments A or B. But P-channel transistor T43 is on, and therefore the signal on line segment A is applied to the input terminal of buffer 41. This buffer is further described by Young in U.S. Patent 5,517,135, incorporated herein by reference.

RAM Block

Returning to Fig. 2, dual port RAM block 13 and connections to dual port RAM memory 131 will now be explained. Dual port RAM block 13 can be connected to the four rows of logic blocks in which RAM block 13 is positioned. Sets of local interconnect lines L0-L3 make programmable connections to logic blocks in the four respective rows spanned by RAM block 13. In addition, RAM 131 can be connected to lines L0-L3 in RAM blocks above or below the one shown.

Programmable connections such as shown by the pentagon symbol in Fig. 4A appear in Fig. 2. As shown by the pentagons, each of the input signals to RAM 131 in Fig. 2 is buffered by a buffer element 41 (Fig. 4A) which is positioned at an intersection of horizontal and vertical lines in RAM block 13 (Fig. 2). Thus the horizontal lines extending to the left edge of RAM 131 are buffered versions of the signals on vertical lines in Fig. 2 to which the horizontal lines connect. Whether the source of a signal driving RAM 131 is above or below the horizontal line depends upon what value is in memory cell M1 (Fig. 4A). Signals which drive input terminals to RAM 131 can be derived from horizontal lines L0 through L3 or they can be derived from RAM blocks above or below the one shown in Fig. 2.

For example, in Fig. 2, in the row of logic blocks accessed by lines L3, line L3-0 can serve as a data-in line and be programmably connected to DINA bus line DIA2. Alternatively, line L3-0 can serve as an address line and be programmably connected to ADDRB line ADDRB2 as indicated by the circle at the intersection of line L3-0 with line ADDRB2. Line ADDRB2 is buffered by a bidirectional buffer to drive line ADB2. Programmable connections represented by pentagons are illustrated in Fig. 4A and discussed above.

Similarly, line L1-15 can serve as a local clock line driven by a logic block in the row accessed by lines L1 and be programmably connected to local clock line LCLK1,

which is connected by a bidirectional buffer to line LCK1. Line LCK1 can in turn be connected to clock input terminal CKB or CKA of either the B or the A port of RAM 131. Programmable connections represented by the triangles at the intersection of line LCK1 and the two clock input lines CKA and CKB to RAM 131 are illustrated in Fig. 3C and discussed above.

Alternatively, line L1-15 can serve as a data-out line and be programmably connected to DOUTB line DOB7.

These and other programmable connections are indicated by circles, triangles and pentagons at the respective intersections.

Pipulation

The term PIP stands for programmable interconnection point. The term pipulation stands for PIP population. Many arrangements of pipulation are possible. Where it is desired to minimize chip area, fewer PIPs are provided (sparser pipulation), and the locations of the PIPs are selected to provide the greatest flexibility and speed for common uses of the structure. If the PIPs are formed from very small devices such as EPROM cells or antifuses, PIPs may be provided at many intersections to increase flexibility. In general, pipulation must be sufficient and the number of local interconnect lines must be sufficient that a user can access all address, data, and control lines used in a configuration. In wide memories, since data lines are not shared, there must be an access point for each data track, as will be discussed in connection with Figs. 6A-6C. When data lines will be used by vertically adjacent RAM blocks, they can not be vertically connected, and sufficient pipulation must be provided to connect the vertical data lines to horizontal interconnect lines in the same RAM block.

Decoder Blocks

Decoder blocks 132 and 133 of Fig. 2 may be formed as shown in Fig. 5. In Fig. 5, decoder block 132 comprises

four two-input multiplexers MUX0 - MUX3. Each multiplexer is controlled by an enable line ENB0 - ENB3 which can in turn be accessed from a line in two of the horizontal lines L0 through L3. In the embodiment of Fig. 5, eight memory cells M1 - M8 provide the inputs to the four two-input multiplexers MUX0 - MUX3. (In another embodiment not shown, the eight multiplexer inputs are routed from neighboring logic blocks as are the four multiplexer control input signals shown in Fig. 2.) The decoder output signal ENB which controls RAM 131 (Fig. 2) is generated by NOR gate NOR1 (Fig. 5) in response to the output signals from the four multiplexers MUX0 through MUX3.

Decoder blocks 132 and 133 allow enable lines in buses ENB and ENA to be used as additional address lines or to be ignored. These blocks can also be used to disable the entire RAM port. The arrangement of Fig. 5 gives great freedom to the FPGA user. Regarding port B, the user can pair two vertically adjacent RAM blocks 13 and use decoder 132 in each block to select between the two of them. Such an arrangement requires only one of the four input lines to decoder block 132. For example, in the two adjacent RAM blocks, multiplexers MUX1 through MUX3 are caused to ignore their respective control signals by loading logical 0's into memory cells M3 through M8. In one of the two RAM blocks, memory cells M1 and M2 are loaded with 01 and in the other, memory cells M1 and M2 are loaded with 10. In this configuration, a logical 0 applied to line ENB0 selects the RAM block configured with 01, and a logical 1 selects the RAM block configured with 10.

Placing 11 into any pair of multiplexer inputs, for example M1 and M2, causes NOR gate NOR1 to provide a logical 0 output signal and thus disables the B RAM block port. This allows the four enable lines to route signals to or from locations above or below the block shown.

Placing 00 into any pair of multiplexer inputs causes that multiplexer to be a don't-care and causes decoder 132

to ignore the signal on the control input of that multiplexer. For example, placing 00 into memory cells M1 and M2 causes multiplexer MUX0 to be a don't-care and allows line ENB0 to be used for routing signals elsewhere while allowing decoder 132 to respond to other enable signals.

In this situation, placing 01 01 01 into the three pairs of memory cells M3-M8 respectively causes decoder 132 to be enabled in response to the address 000 on enable lines ENB1-ENB3. The values 01 in a pair cause the respective multiplexer to forward the control signal to the output. The values 10 in a pair cause the multiplexer to forward the complement of the control signal to the output. Eight RAM blocks 13 can be separately addressed by placing different permutations of 0 and 1 into the three pairs of multiplexer inputs in memory cells M3-M8 of the eight RAM blocks 13 and placing the address on lines ENB1-ENB3. And of course, sixteen RAM blocks can be addressed by using all four enable lines.

A default value of 11 11 11 11 loaded into the eight memory cells M1-M8 disables the RAM so that no power is consumed and no contention occurs in response to any configuration of other parts of RAM block 13.

Although Fig. 5 illustrates a NOR gate for generating the control output signal ENB, an OR, NAND or AND gate can be used just as well, with different values in memory cells M1-M8 achieving the desired result. For example, if an AND gate were used, the default value would be 00 00 00 00. Different values on enable lines ENB1-ENB3 would also be used to produce the desired result.

In one embodiment, a disabling decoder output signal (for example a logical 0 from NOR gate NOR1) also causes all output signals from that port (for example all signals from RAM 131 which drive DOUTB bus lines DOB0 - DOB8) to be tristated. This allows lines DOB0 - DOB8 to be driven by another RAM 131 above or below the one shown. However, tristating also makes it possible for lines DOB0 - DOB8 (and other data out lines) to be left floating when no RAM

output signals are driving the lines. Thus small keeper circuits or pull-up resistors are attached to these lines.

Fig. 4B

If several copies of Fig. 1B or Fig. 2 are placed vertically adjacent to each other, the lines or circles at the top of one copy will join straight lines at the bottom of another copy. Line B in the symbol of Fig. 4B is in the upper copy (or block) and line A and the circle in the symbol of Fig. 4B are in the lower copy. The combination of a triangle and an open circle with a line to its center shown at the left of Fig. 4B indicates a bidirectional buffer with default input shown at the right of Fig. 4B. This is the same circuit shown in Fig. 4A, and is not explained again. Part of the symbol of Fig. 4B appears in Figs. 6A-6C, discussed below. The symbol of Fig. 4B also appears in Figs. 7A-7C below.

RAM Block Grouping and Memory Length/Width Control

Fig. 6A illustrates a RAM block group 101 including four adjacent RAM blocks 13-5 through 13-8, each structured as shown in Fig. 2. Other numbers of RAM blocks can also be used together. RAM block group 101 is also illustrated in Fig. 1A. Each RAM block spans four logic blocks so the four RAM blocks in group 101 span 16 logic blocks. The four adjacent blocks can be configured as a 1024x9 RAM, a 512x18 RAM, or a 256x36 RAM. Each RAM block includes a RAM 256x9, that is, a RAM having a depth of 256 words, each word having a width of 9 bits. The 9-bit word is used in this embodiment instead of an 8-bit word because many users want a parity bit, and getting a parity bit with only 8-bit words is awkward. Providing the 9th bit when unused is not so expensive since the 9th bit shares the decode logic of the other 8 bits and the 12% increase in memory area produces a small increase in total area. In this embodiment, width (word length) and length (number of addresses) depend on how you turn on the bidirectional buffers and pass gates between adjacent RAM

blocks 13. Figs. 6B and 6C illustrate alternative configurations of RAM block group 101.

In Fig. 6A, the four RAM blocks 13-5 through 13-8 are configured as a 1024x9 memory. For 1024x9, there must be 10 address bits. The ADDR bus (buses ADDRA and ADDRB in Fig. 2) has 8 bits. As shown in Fig. 6A, the four adjacent ADDR buses in the four adjacent RAM blocks 13-5 through 13-8 are connected together. The eight address bits in the ADDR bus (or 16 on dual address buses ADDRA and ADDRB of Fig. 2) address the 256 words (2^8) inside RAMs 131. Since four locations, one in each RAM 131 are addressed by the same 8-bit address on BUS ADDR, two enable lines EN (ENA or ENB in Fig. 2) are also used as address bits and connected together to provide the 9th and 10th address bits as discussed above in connection with Fig. 5. The data-in and data-out lines in one RAM block are connected to the corresponding data-in and data-out lines in the adjacent RAM blocks since only one RAM in the four RAM blocks will be addressed at any one time. The data placed onto the data-out line may be picked up at a local horizontal line at several places along the four RAM blocks (or 16 logic blocks). The write enable WE and local clock LCLK lines are also connected together to group the four RAM blocks together. Black dots indicate buses having lines in adjacent blocks which have been connected through bidirectional buffers or pass gates, and clear circles indicate lines which have not been connected through bidirectional buffers or pass gates. Note at the top and bottom of Fig. 6A, the bidirectional buffers and pass gates show disconnected lines from the next RAM block above, whereas interior to the group of four, all pairs of lines in adjacent RAM blocks have been connected.

As shown in Fig. 6B, to form a 512x18 memory, the data lines of the four RAM blocks are divided into two groups as indicated by the open circles PIN2 and POUT2 while the address, enable, write-enable, and local clock lines remain together. Only one of the enable lines is used as an address line, the others being ignored by the

don't-care configuration of the decoder discussed above in connection with Fig. 5. So two RAM cells, for example one RAM cell in RAM block 13-5 and one RAM cell in RAM block 13-7, are accessed in response to one 9-bit address. For reading, the two addressed RAM cells each place their 9 bits of data onto the data-out buses DOUT simultaneously. For writing, the two addressed RAM cells each take their 9 bits of data from the data-in buses DIN simultaneously. The data-in and data-out lines are split at the middle at PIN2 and POUT2 into two groups. Horizontal data lines in each of the upper and lower groups read and write data, and since the data lines are split there is no contention. The data bits stored in the upper two RAM blocks must be detected on horizontal lines connected to the upper two RAM blocks 13-5 and 13-6 and the lower bits must be detected on the lower lines connected to RAM blocks 13-7 and 13-8. In this example, addresses may come from logic block locations adjacent to any of the four RAM blocks or even from more distant locations which can be connected to the horizontal lines shown. In another example, bus ADDR in RAM block 13-5 may be connected to the address bus above (not shown), in which case the address may be derived from logic blocks even farther from the addressed RAM blocks. Thus in response to one 9-bit address, 18 data bits are written or read. 512 distinct locations are addressed by the 9-bit address, so the memory is 512x18.

As shown in Fig. 6C, for a 256x36 memory, the data input and output lines of the four RAM blocks are separated into four groups while the address lines ADDR are connected together. In this configuration, only 8 address lines are used. The enable lines are not used for addressing. Thus four RAM blocks place a data word onto a data-out bus or read a data word from a data-in bus in response to a single address. The four sets of data must then be detected or provided by logic blocks in the regions of the respective RAM blocks.

Dual port RAM Configurations

Since each 256x9 RAM is a dual port RAM, it can be configured as two separate single-port 128x9 RAMs, or as one single-port 128x18 RAM. Recall that all 256 x 9 = 2304 RAM bits are addressable from both address buses.

To form two 128x9 RAMs, one of the eight address bits, for example the most significant address bit, is permanently set to one state on the address bus for one port and the other state for the other port. This causes half the RAM to be addressed by the seven remaining bits on one address bus and half by the seven bits on the other address bus. The least significant seven address lines of one address port are connected to one set of seven vertical address lines and the least significant seven address lines of the other address port are connected to another set of seven vertical address lines. The data-in ports are connected to separate data-in lines and the data-out ports are connected to separate data-out lines.

To form one single-port 128x18 RAM, the most significant bit in one address port is connected to the enabling voltage source, for example logical 1 and the most significant bit in the other address port is connected to ground. The remaining seven address lines from the two ports are connected to the same seven vertical address lines. Thus any address on the seven vertical address lines addresses two sets of RAM cells. The eighteen vertical data-out lines are connected to the eighteen data-out lines of the two ports and thus receive the contents of the two sets of addressed RAM cells, which then comprise a single 18-bit word.

Second Embodiment, Figs. 7A-7C

A second embodiment of the RAM block of the invention is shown in Figs. 7A-7C. The RAM block of Fig. 7A includes RAM 531 and lines for connecting RAM 531 to other portions of the chip. RAM 531 is configurable to four different configurations: 4096x1, 2048x2, 1024x4, and 512x8. Horizontal lines extend left from RAM 531. These

provide address, data-in, and control signals to RAM 531. Horizontal lines extending to the right from RAM 531 receive data-out signals from RAM 531. As in Fig. 2, dedicated vertical lines extending through the RAM block of Fig. 7A provide the interface between logic blocks of the FPGA chip and the RAMs 531 in the RAM blocks of the chip. As in Fig. 2, programmable connectors at the top of the figure allow the RAM block of Fig. 7A to connect to an identical RAM block above. The internal configurations of RAM 531 may be controlled in a manner similar to that discussed above in connection with Figs. 6A-6C, by well-known methods, or by the novel structure described by Nance et al. in co-pending application serial no. 08/687,902 filed July 29, 1996 [X-264-1N].

As in Fig. 2, four groups of horizontal lines in sets L0 through L3 extend through the RAM block of Fig. 7A. These lines are part of the general interconnect structure for the logic blocks of the FPGA chip. Programmable connectors allow these horizontal lines to be connected to the vertically extending address, data-in, data-out, and control lines. In the embodiment of Fig. 7A, longer horizontal lines HL0 through HL3 carry data-out signals to logic blocks farther away. The unlabeled triangles at selected intersections of horizontal and vertical lines allow programmable connections at the intersections. As discussed in connection with Fig. 4B, the combination of a triangle and an open circle with a line to its center indicates a bidirectional buffer with default input. In Fig. 4B or Figs. 7A-7C, the circle can be thought of as a programmable buffered connection to a RAM block above or below the figure. The triangle (with circle above) can be thought of as a programmable buffered connection into RAM 531.

Difference Between Embodiments

There is a basic difference between the embodiments of Fig. 2 and Figs. 7A-7C. In Fig. 2, data-out lines from several RAMs are connected to the same vertical data-out

lines, and in a deep configuration (1024x9 shown in Fig. 6A for example) several RAM blocks may be connected to the same line. Contention is avoided by using the enable lines to avoid placing more than one signal onto a data-out line at one time.

In the embodiment of Figs. 7A-7C, there are no enable lines. Contention is avoided by staggering the data-in and data-out lines. At the tops of Figs. 7A-7C, programmable connectors, such as DIC31 and DIC30 labeled in Fig. 7B, connect to lines extending into the RAM block above. Looking at Fig. 7B, data-in line DI30 connects through programmable data-in connector DIC30 to data-in line DI31', which continues as line DI31 in the RAM block above. Left-most line DI31 connects through programmable data-in connector DIC31 to line DI16', which continues as line DI16 in the RAM block above. Other lines are staggered by lines above their respective programmable connectors to move left in the RAM block above. The same staggering arrangement occurs for the data-out lines to the right of RAM 531. Looking at both Figs. 7A and 7B, one can see that the top line L3-23 (Fig. 7A) of lines L3 can be programmably connected to vertical line DI31 and programmably connected through data-in PIP DIP31 (Fig. 7B) to data-in line DIB3 in RAM 531. Alternatively or additionally, the RAM above RAM 531 may be used by setting programmable connector DIC31 to conduct signals upward, thereby connecting line DI31 to line DI16', which continues as line DI16 in the RAM block above. Line DI16 can be seen in Fig. 7B to connect through PIP DIP16 to data-in line DIB7. Thus line DI31 can connect to data-in line DIB3 in the RAM block shown or to line DIB7 in the RAM block above.

Separation of staggered lines for maximum flexibility in all RAM configurations

Note the numbering of the data-in and data-out lines in Figs. 7B and 7C. The data-in and data-out lines in RAM 531 are numbered according to which bit is most

significant, bit 7 being most significant and bit 0 being least significant. The numbering is not consecutive. (Of course the illustration could have been drawn with consecutive numbering and a correspondingly different pattern of PIPs external to RAM 531.) The PIPs and staggered vertical lines are arranged so that x1, x2, x4, and x8 RAM memories can all be accessed through the vertical lines with no contention over the largest possible span of vertically adjacent RAM blocks for the number of vertical data lines provided. For a x1 RAM, that is, when RAM 531 has been configured with one bit in a data word, only the D0 line is used (for example lines DIA0, DIB0, DOA0, and DOB0) and the vertical line which connects to a particular horizontal D0 line does not connect to that same vertical line until 16 RAM blocks have passed. For a x2 RAM, that is, when RAM 531 has been configured with 2 bits in a data word, only lines D0 and D1 are used. These are separated from each other by the staggering structure at the top of Figs. 7A-7C so that a D0 line does not conflict with a D1 line until 8 RAM blocks have passed. Similarly, for a x4 RAM, lines D0, D1, D2, and D3 are used and don't conflict until four RAM blocks have passed. Finally, for the x8 configuration, all the horizontal data lines are used. Since there are 8 horizontal data lines and 16 vertical data lines, the configuration can be arranged so that even when eight data lines are used there is no conflict until 2 RAM blocks have passed.

In the embodiment of Fig. 7A, a x1 memory is assumed to use only the 0th data ports. However, in another embodiment, a RAM block in a x1 memory configuration can place a single signal onto several or all data ports, and thereby increase the routing flexibility without increasing the space required for pipulation. Similarly, a x2 configuration can use more than two data ports and a x4 configuration can use more than four data ports.

Pipulation

Looking at Fig. 7A, a regular pattern of PIPs allows interconnect lines L0 through L3 to access RAM block 531 as well as other RAM blocks above and below the figure. Lines in sets L0 through L3 can serve as address or data lines. For example, line L3-23 in set L3 can be used as an address line by turning on the PIP at the intersection of line L3-23 and vertical address line A23. Vertical address line A23 can in turn access horizontal A-port address line ADA11 or horizontal B-port address line ADB11 by turning on the PIP at the intersection of those two lines. Vertical address line A23 can be connected to another vertical address line A23 in the RAM block above the figure by turning on connector AC23 at the top of the figure and this vertical address line can in turn be connected to horizontal address lines such as shown in Fig. 7A. Thus, line L3-23 can access many different horizontal address lines in many different RAM blocks. Each of the lines L3-0 through L3-23 in set L3 can access one of the horizontal address lines leading to RAM 531.

The PIPs for connecting set L2 to vertical address bus ADDR are offset from corresponding PIPs for set L3. PIPs for sets L1 and L0 are likewise offset. The four sets L0 through L3 can each access all horizontal address lines ADA11-ADA0 in the A-port and all horizontal address lines ADB11-ADB0 in the B-port through vertical address bus ADDR, though different vertical address lines are used by the four sets of lines L0-L3 because the offset of PIPs is different for the four sets of lines L0-L3. For example, in set L3, line L3-23 can access horizontal address lines ADA11, ADA10, ADB4 and ADB5. In set L2, line L2-23 can access the same horizontal address lines as line L3-23 but through vertical address line A22. Lines L2-23 and L1-23 access horizontal address lines ADA10, ADA9, ADB4 and ADB3 through vertical address lines A21 and A20 respectively.

Lines in sets L0-L3 can also serve as data-in or data-out lines. For example, line L3-23 in set L3 can be

used as a data-in line by turning on the PIP at the intersection of line L3-23 and vertical data-in line DI31 and turning on one or more PIPs at the intersections of vertical data-in line DI31 and horizontal data-in lines DIB3 and DIA3 (see labeling in Fig. 7B).

Note that line L3-23 can not serve as a data-out line from the B-port of RAM 531 in Fig. 7A. Though a PIP is present at the intersection of line L3-23 and vertical data-out line DO27, no PIP is present at the intersection of vertical data-out line DO27 and any horizontal data-out lines from the B-port of RAM 531. However, line L3-23 can serve as a data-out line from the B-port of the corresponding RAM 531 above Fig. 7A. Turning on connector DOC31 (see Fig. 7C) connects vertical data-out line DO31 to data-out line DO16', which continues as data-out line DO16 in the RAM block above the figure. As can be seen in Fig. 7C, vertical data-out line DO16 can receive data from horizontal data-out line DOB0 by turning on the PIP at that intersection. Other patterns of PIPs allow other options. As discussed above, a more dense pipulation gives more options but requires more chip area to implement.

Dual-Port RAM Feature

With the separate data-in and data-out lines, some logic blocks can read RAM 531 through port B while other logic blocks are writing to RAM 531 through port A. Notice that 24 vertical address lines A0-A23 are provided and that in a 4096-deep configuration 12 address lines are used. For example, if the odd-numbered ones of address lines A0-A23 are used for writing to data-in port A, then the even numbered address lines can be used for reading from data-out port B. (Any mixture of lines can be used for ports A and B as long as there are sufficient lines and sufficient PIPs.)

For writing to port A and reading from port B in the 4096x4 RAM, four data bits will be written to vertical data-in lines DI0 through DI3 by turning on four pairs of

PIPs. One PIP will be turned on at the intersection of horizontal line L3-20 and vertical data-in line DI19, a second PIP at the intersection of horizontal line L2-20 and vertical data-in line DI18, a third at the intersection of horizontal line L1-20 and vertical data-in line DI17, and a fourth at the intersection of horizontal line L0-20 and vertical data-in line DI16.

To complete the paths, PIP DIP19 in RAM 531 is turned on to connect line L3-20 to data-in port DIB0. Thus, the signal on line L3-20 is received on line DIA0 in RAM 531, and written by well-known means to the addressed memory cell in RAM 531. PIP DIP18 is not turned on. Thus, the signal on line L2-20 is not connected to line DIA0 in RAM 531. However, in an identical RAM block above, PIP DIP19 is turned on, thus the signal on line L2-12 is received on line DIA0 in the RAM above RAM 531. The signal on line L1-20 which is placed on vertical data-in line DI17 is connected by connector DIC17 to vertical data-in line DI18 in the RAM block above Fig. 7A and further through a connector DIC18 in the RAM block above Fig. 7A to a vertical data-in line DI19 in the RAM block two above Fig. 7A. From there it connects through a PIP DIP19 to a horizontal data-in line DIA0 and into the RAM two above Fig. 7A. Similarly, the signal on line L0-20 which is placed on line DI16 in Fig. 7A is connected to a horizontal data-in line DIA0 in the RAM block three above Fig. 7A.

Thus four bits of a data word which are placed onto lines L3-20, L2-20, L1-20, and L0-20 by logic blocks adjacent the RAM block of Fig. 7A are written into memory cells in four different RAM blocks, three of which are above Fig. 7A.

In each of four RAM blocks, only horizontal data-out line DOB0 is used. This results in four data-out signals being placed onto vertical data-out lines DO16 through DO19 of Fig. 7C or Fig. 7A. Although PIPs are present at the intersections of these four lines and horizontal lines L3-20, L2-20, L1-20, and L0-20, these four horizontal

lines can not be used for reading because they are already being used for the writing operation described above. However, PIPs are also present at intersections with lines L3-14, L2-14, L1-14, and L0-14 and these lines can be used for writing by turning on the respective PIPs.

Example Applications of RAM Block of Figs. 7A-7C

RAM 531 is a dual-port RAM with four configurations. RAM 531 contains 4096 memory cells, which can be configured by well known internal structures as a 4096x1 RAM, a 2048x2 RAM, a 1024x4 RAM or a 512x8 RAM. There are 24 horizontal address lines, 12 for each port, in the embodiment of Fig. 7A.

In a 4096x1 configuration, 12 horizontal address lines address each port (2^{12} = 4096). In the 4096x1 configuration, only horizontal data-in lineS DIA0 AND DIB0 and horizontal data-out lineS DOA0 and DOB0 are used.

In a 2048x2 RAM configuration, 11 horizontal address lines address each port (2^{11} = 2048), leaving the most significant address bit for each port unused. Two data-in lines for each port DIA0, DIA1, DIB0, and DIB1 and two data-out lines for each port DOA0, DOA1, DOB0 and DOB1 are used.

In a 512x8 RAM, nine address lines for each port, lines ADB8-ADB0 and ADA8-ADA0 are used for addressing. The three most significant address lines are ignored. In the 512x8 configuration, all 8 data lines for each port are used.

Fig. 8A shows an example application of the embodiment of Figs. 7A-7C when the RAM blocks are in the 512x8 configuration. Four RAM blocks are used in conjunction. They are configured as a 512x32 RAM. Since the logic that accesses the memory cells in the four adjacent RAM blocks has been placed so that each logic block accesses an adjacent RAM block, the vertical connections from one RAM block to the next are not needed or used. Fig. 8A shows the vertical data-out routing lines within one RAM block being used to carry signals

from RAM block output ports to logic block input ports.

Memory cells in RAM block 531-8 store the least significant bits 0 through 7 of each 32-bit word. The output ports DOB0 through DOB7 of RAM block 531-8 are connected to logic blocks in the four rows adjacent to RAM block 531-8 by turning on PIPs to the vertical data-out lines. For simplicity, only the PIP at the intersection of vertical data-out line DO-16 and one of the horizontal lines is labeled. In the example shown, data-out port DOB0 in RAM block 531-8 is connected to vertical data-out line DO-16 by turning on the PIP at the intersection of these two lines. By turning on the PIP at the intersection of vertical line DO-16 and horizontal line L0-5, data-out port DOB0 is connected to line L0-5 to provide the 0 data bit to logic block 25-40. Similarly, data-out port DOB1 is connected to vertical line DO-22, which is in turn connected to line L0-7 of logic block 25-40. The other 30 data bits of the 32-bit word are connected as shown. Additional connections not shown in Fig. 8A allow logic blocks to access vertical address lines and data-in lines as well as to apply clock and other required signals to implement the desired memory functions.

Deeper RAMs that Use Connections Between Vertical Lines

With the dedicated vertical lines and PIP pattern shown, larger RAMs are conveniently formed by connecting the vertical address, data, and control lines together. Fig. 8B illustrates use of the programmable interconnections between vertical data-out lines in adjacent RAM blocks to implement a 2048x8 RAM memory in four adjacent RAM blocks. Fig. 8B also shows how the data-out lines are connected to the logic blocks for this configuration. Logic blocks 25-33 through 25-36 receive the eight bits 0 through 7 of a data word. These four logic blocks are adjacent RAM block 531-7. But some data bits are stored in RAM blocks that are not adjacent logic blocks 25-33 through 25-36. Thus the vertical lines are

used to conveniently make the transfer. For example, memory bit 2 provided to logic block 25-35 is stored in RAM block 531-8 and accessed through port DOB0. Routing is connected by turning on the PIP at the intersection of DOB0 and DO-16 adjacent RAM block 531-8, turning on the PIP at the intersection of DO-17 and L1-5 adjacent RAM block 531-7, and turning on tristate buffer TB-1, which connects vertical line DO-16 adjacent RAM block 531-8 to vertical line DO-17 adjacent RAM block 531-7. To supply bits 0 and 1 does not require that tristate buffers be used. To supply bits 2, 3, 6, and 7 requires that one tristate buffer be turned on for each bit. To supply bits 4 and 5 requires that two tristate buffers be turned on for each bit. For example, bit 5, which is supplied on output port DOB1 of RAM block 531-5 is routed to its destination in logic block 25-34 by turning on tristate buffers TB2 and TB3 as well as the PIP at the intersection of DOB1 and DO-24 adjacent RAM block 531-5 and the PIP at the intersection of DO-22 and line L2-6 adjacent RAM block 531-7.

RAMs Using RAM Blocks and Logic

Additional flexibility in forming RAMs, especially large RAMs, is achieved by combining the logic and routing structures of the FPGA with the RAM blocks.

Fig. 8C shows an 8192x2 RAM formed using two logic blocks for multiplexing signals and two additional logic blocks for routing signals. Fig. 8C shows the address and data-in lines as well as the data-out lines shown in Figs. 8A and 8B.

Since the embodiment of Fig. 7A uses 4096 memory cells in a RAM block, it takes 2 RAM blocks to store one bit of all words. In Fig. 8C, bit 0 is stored in RAM blocks 531-7 and 531-8 while bit 1 is stored in RAM blocks 531-5 and 531-6. Port B has been selected to access the memory, for both reading and writing. Writing is to port DIB0 and reading is from port DOB0. A memory 8192 cells deep requires 13 address lines. Port B has only 12

address lines. Therefore, twelve address signals A0 through A11 are applied to the 12 address ports. Ten of the 12 address signals are applied by turning on one PIP between a logic block and a vertical address line and one PIP between a vertical address line and an address port. Two of the address signals A8 and A11 are applied to corresponding address ports by routing the address signal to an adjacent logic block and from there to a vertical address line and on to an address port. For example, address signal A11 is applied to local interconnect line I1, and by turning on PIP9 to horizontal line L2-15, which is connected by turning on PIP10 to vertical interconnect line ADDR14. PIPs then connect this vertical interconnect line to address port ADB11 in each of RAM blocks 531-5 through 531-8.

The thirteenth address signal A12 is logically combined with the write enable signal DWE so that only the addressed memory cells are written. This logical combination can be done several ways. In Fig. 8C, logic blocks 24-31 and 24-32 do the logical combination of the write-enable and address signals. Logic block 24-31 generates the AND function of DWE and A12. The AND function output is applied to the write-enable ports WEB of RAM blocks 531-5 and 531-7 so that when DWE is high (data to be written) and A12 is high (the address is in the upper half of its range), a write enable signal will be applied to RAM blocks 531-5 and 531-7. Logic block 24-32 generates the AND function of DWE and the inverse of A12 so that when an address in the lower half of its range is to be written, RAM blocks 531-6 and 531-8 are enabled.

On the data-out side, logical combination with address signal A12 is performed in logic blocks 25-35 and 25-36, which are each configured as multiplexers. Data bit DO1 is generated in logic block 25-35. Logic block 25-35 receives the DOB0 output signals from RAM blocks 531-6 and 531-8. When A12 is high, the multiplexer in logic block 25-35 selects the data bit from RAM block 531-6 and when A12 is low, the multiplexer in logic block 25-

35 selects the data bit from RAM block 531-8. A similar situation exists in logic block 25-36. Thus the multiplexer outputs from logic blocks 25-35 and 25-36 are the two bits DO0 and DO1 of the addressed word in the 8192-deep memory.

Note that when PIPs in the vertical lines of the RAM block were not available to connect an output signal from RAM block 531-5 to logic block 25-35, local interconnect, which is typically provided in FPGA logic blocks, carries the signal. In this example, data-out port DOB0 of RAM block 531-5 is connected to logic block 25-35 by turning on PIP1, TB5, TB4, PIP2, PIP3, and PIP4. Of these, PIP3 and PIP4 are part of the logic block local interconnect structure such as found in conventional FPGAs.

Address line A12 is routed to the multiplexer control lines of logic blocks 25-35 and 25-36 through general interconnect structure by turning on PIP5 through PIP8.

Many routing lines and interconnect structures not shown in Fig. 8C but well known in the prior art would of course be provided in a commercial embodiment of an FPGA. Since showing such structures is believed to obscure the understanding of the present invention, they have not been shown here.

Larger RAMs

It is possible to form a 4096x16 RAM by joining together 16 RAM blocks in a vertical column before there will be any contention on the vertical address, data-in, and data-out lines. Larger memories can be formed by segmenting RAM blocks in the same vertical column, by using RAM blocks from more than one column, or by using vertical general interconnect (well known in FPGA architectures) for interconnecting or accessing portions of the RAM. As shown in Fig. 1A, an FPGA chip typically includes multiple columns of RAM blocks. Horizontal lines such as L0-L3 can be used to access more than one column of RAM blocks. Wider RAMs are formed by connecting together more RAM blocks in a vertical column.

Other Embodiments

The embodiment of Fig. 7A shows 24 address lines, and 16 data-in and data-out lines for each port. Thus 16 vertically adjacent RAM blocks can conveniently be combined, that is, used in combination without producing contention on data-out lines or ambiguity on input lines. Other numbers of lines, other patterns of PIPs, and other sizes of RAMs can of course be used.

CLAIMS

1. An FPGA with RAM comprising:
a plurality of logic blocks arranged in rows and columns;
a plurality of RAM blocks arranged in columns, said RAM blocks having address ports and data ports;
an interconnect structure comprising conductive lines arranged in rows;
a set of vertical lines, each set associated with a column of said RAM blocks;
means for connecting said logic blocks to said interconnect structure;
means for connecting said vertical lines to said interconnect structure; and
means for connecting said address and data ports to said vertical lines.

2. An FPGA with RAM as in Claim 1 further comprising:
means for segmenting said vertical lines.

3. An FPGA with RAM as in Claim 2 wherein some of said means for segmenting are bidirectional buffers.

4. An FPGA with RAM as in Claim 2 wherein some of said means for segmenting are pass transistors.

5. An FPGA with RAM as in Claim 3 wherein said bidirectional buffers segment a line into line segments, each segment being adjacent to a RAM block.

6. An FPGA with RAM as in Claim 1 wherein some of said RAM blocks comprises dual port RAMs having two address ports and at least two data ports, all ports being accessible by said vertical lines.

7. An FPGA with RAM as in Claim 1 wherein each of said RAM blocks can be configured to a plurality of length-to-width ratios.

8. An FPGA with RAM as in Claim 1 wherein each of said RAM blocks spans four logic blocks.

9. An FPGA with RAM as in Claim 2 wherein a first one of said RAM blocks is connectable to different ones of said conductive lines from a second one of said RAM blocks vertically adjacent to said first RAM block.

10. An FPGA with RAM as in Claim 9 wherein a vertical line in said first RAM block can be connected to a different vertical line in said second RAM block such that a signal from a given port in said first RAM block does not conflict with a signal from the same given port in said second RAM block.

11. An FPGA with RAM blocks comprising:
a plurality of logic blocks arranged in rows and columns;
a plurality of general interconnect lines [L0-L3];
a plurality of RAM blocks [13] each comprising:
a plurality of RAMs [131], each being addressed by an address bus [ADDRA, ADDRB] and accessed by at least one data bus [DINA, DINB, DOUTA, DOUTB], and being enabled by at least one enable line [WEA, WEB, ENA, ENB]; and
means [PIPs] for connecting each line in each of said address bus and said data bus to at least one of said general interconnect lines; and
means for programmably connecting each enable line, each line in each of said address bus and said at least one data bus in one of said RAM blocks to a corresponding line in another of said RAM

blocks, whereby a larger RAM is formed having a programmable configuration;

said data bus in one of said RAM blocks being connected to a data bus in another of said RAM blocks through a tristatable bidirectional connector.

12. An FPGA as in Claim 11 in which said means for programmably connecting each enable line, each line in each of said address bus and said at least one data bus in one of said RAM blocks to a corresponding line in another of said RAM blocks comprises a bidirectional buffer for making each programmable connection.

13. An FPGA as in Claim 11 in which said means for programmably connecting each enable line, each line in each of said address bus and said at least one data bus in one of said RAM blocks to a corresponding line in another of said RAM blocks comprises a pass gate for making each programmable connection.

14. An FPGA as in Claim 11 in which said means for programmably connecting each enable line, each line in each of said address bus and said at least one data bus in one of said RAM blocks to a corresponding line in another of said RAM blocks comprises at least one bidirectional buffer for making a corresponding at least one programmable connection, and at least one pass transistor for making a corresponding at least one programmable connection.

15. An FPGA as in Claim 14 in which said at least one bidirectional buffer comprises bidirectional buffers for connecting all lines which provide input signals to one of said RAMs and said at least one pass transistor comprises pass transistors for connecting all lines which provide output from one of said RAMs.

16. An FPGA as in Claim 11 in which said at least one enable line is a plurality of enable lines providing input signals to a decoder [132].

17. An FPGA as in Claim 16 in which said decoder comprises:

a plurality of multiplexers [MUX0-MUX3] each controlled by one of said enable lines, said multiplexers each receiving a plurality of multiplexer input signals having a selectable value.

18. An FPGA as in Claim 17 in which said multiplexer input signals are each stored in a memory cell.

AMENDED CLAIMS

[received by the International Bureau on 30 December 1997 (30.12.97);
original claims 1,6 and 9-11 amended; remaining claims unchanged
(4 pages)]

1. An FPGA with RAM comprising:
a plurality of logic blocks arranged in rows and columns;
a plurality of RAM blocks dedicated to the RAM function and arranged in columns, said RAM blocks having address ports and data ports;
an interconnect structure comprising conductive lines arranged in rows;
a set of vertical lines, each set programmably driving said address ports of a column of more than one of said RAM blocks;
means for connecting said logic blocks to said interconnect structure;
means for connecting said vertical lines to said interconnect structure; and
means for connecting said address ports to said vertical lines.

2. An FPGA with RAM as in Claim 1 further comprising:
means for segmenting said vertical lines.

3. An FPGA with RAM as in Claim 2 wherein some of said means for segmenting are bidirectional buffers.

4. An FPGA with RAM as in Claim 2 wherein some of said means for segmenting are pass transistors.

5. An FPGA with RAM as in Claim 3 wherein said bidirectional buffers segment a line into line segments, each segment being adjacent to a RAM block.

6. An FPGA with RAM as in Claim 1 wherein some of said RAM blocks comprise dual port RAMs having two address ports and at least two data ports, all such address ports being accessible by said vertical lines.

7. An FPGA with RAM as in Claim 1 wherein each of said RAM blocks can be configured to a plurality of length-to-width ratios.

8. An FPGA with RAM as in Claim 1 wherein each of said RAM blocks spans four logic blocks.

9. An FPGA with RAM as in Claim 2, further comprising:
a group of data lines running vertically, each group programmably driving said data ports of a column of more than one of said RAM blocks;
means for connecting said data lines to said interconnect structure;
means for connecting said data ports to said data lines; and
means for segmenting said data lines;
wherein a first one of said RAM blocks is connectable to different ones of said data lines from a second one of said RAM blocks vertically adjacent to said first RAM block.

10. An FPGA with RAM as in Claim 9 wherein a first one of said group of data lines driving said first RAM block can be connected without utilizing said interconnect structure to a different one of said group of data lines driving said second RAM block such that a signal driving a given data port in said first RAM block does not conflict with a signal driving the same given data port in said second RAM block.

11. An FPGA with RAM blocks comprising:
a plurality of logic blocks arranged in rows and columns;
a plurality of general interconnect lines [L0-L3];
a plurality of RAM blocks [13] each comprising:
a plurality of RAMs [131] dedicated to the RAM function, each being addressed by an

address bus [ADDRA, ADDRB] and accessed by at least one data bus [DINA, DINB, DOUTA, DOUTB], and being enabled by at least one enable line [WEA, WEB, ENA, ENB]; and

means [PIPs] for connecting each line in each of said address bus and said data bus to at least one of said general interconnect lines; and

means for programmably connecting each enable line, each line in each of said address bus and said at least one data bus in one of said RAM blocks to a corresponding line in another of said RAM blocks, whereby a larger RAM is formed having a programmable configuration;

said data bus in one of said RAM blocks being connected to a data bus in another of said RAM blocks through a tristatable bidirectional connector.

12. An FPGA as in Claim 11 in which said means for programmably connecting each enable line, each line in each of said address bus and said at least one data bus in one of said RAM blocks to a corresponding line in another of said RAM blocks comprises a bidirectional buffer for making each programmable connection.

13. An FPGA as in Claim 11 in which said means for programmably connecting each enable line, each line in each of said address bus and said at least one data bus in one of said RAM blocks to a corresponding line in another of said RAM blocks comprises a pass gate for making each programmable connection.

14. An FPGA as in Claim 11 in which said means for programmably connecting each enable line, each line in each of said address bus and said at least one data bus in one of said RAM blocks to a corresponding line in another

of said RAM blocks comprises at least one bidirectional buffer for making a corresponding at least one programmable connection, and at least one pass transistor for making a corresponding at least one programmable connection.

15. An FPGA as in Claim 14 in which said at least one bidirectional buffer comprises bidirectional buffers for connecting all lines which provide input signals to one of said RAMs and said at least one pass transistor comprises pass transistors for connecting all lines which provide output from one of said RAMs.

16. An FPGA as in Claim 11 in which said at least one enable line is a plurality of enable lines providing input signals to a decoder [132].

17. An FPGA as in Claim 16 in which said decoder comprises:

a plurality of multiplexers [MUX0-MUX3] each controlled by one of said enable lines, said multiplexers each receiving a plurality of multiplexer input signals having a selectable value.

18. An FPGA as in Claim 17 in which said multiplexer input signals are each stored in a memory cell.

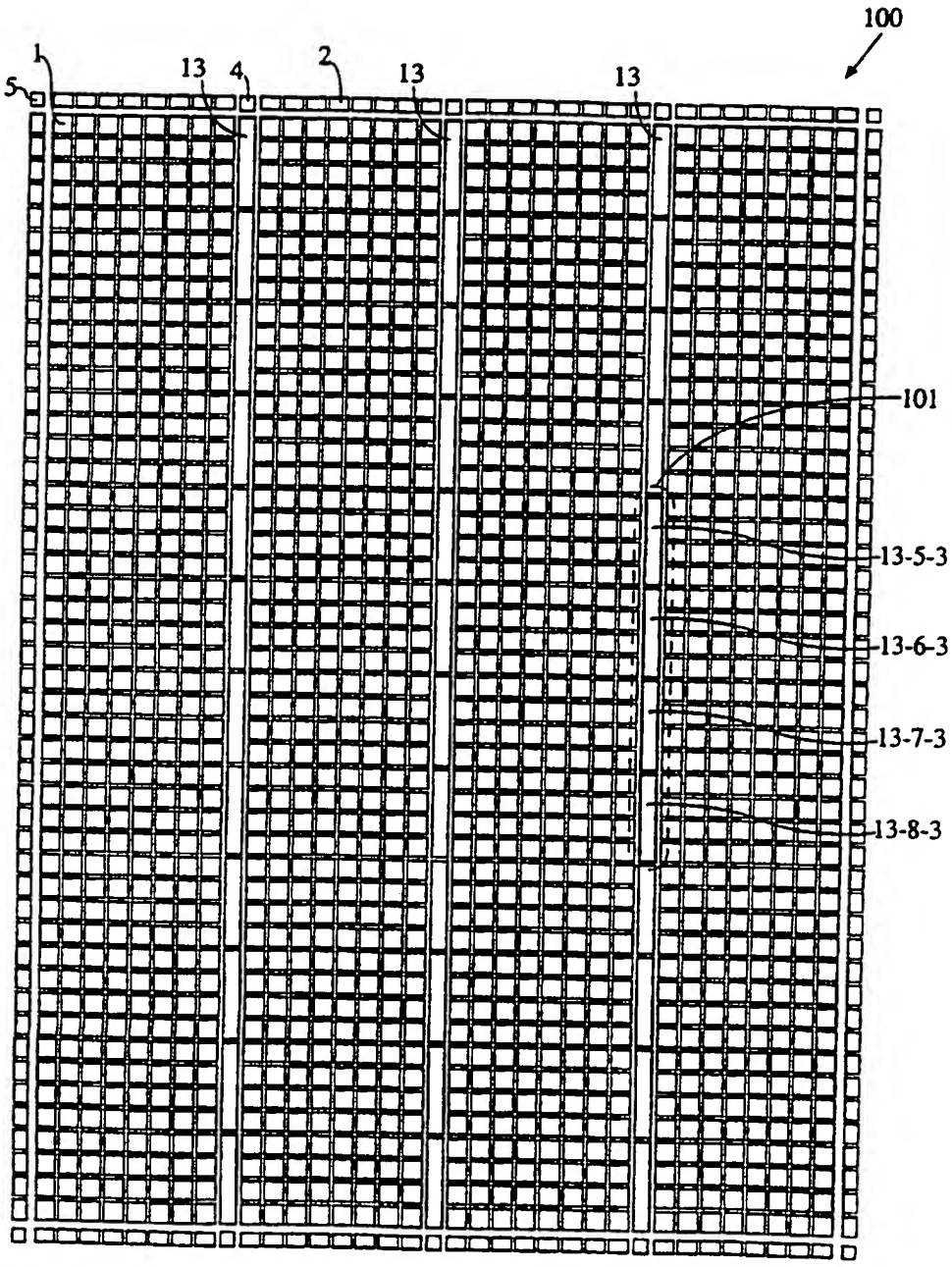


FIG. 1A

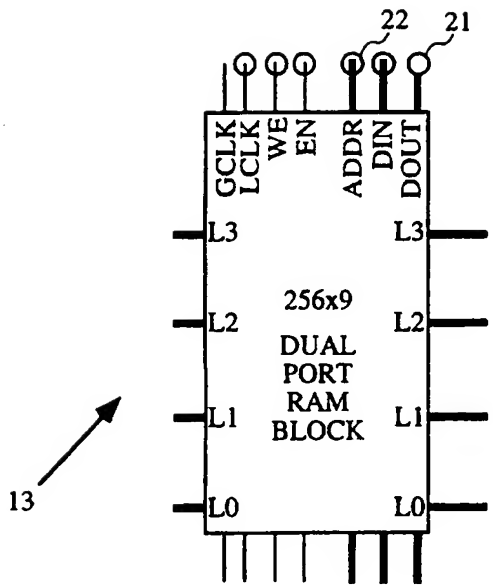

22
21
GCLK
LCLK
WE
EN
ADDR
DIN
DOUT
L3
L2
L1
L0
256x9
DUAL
PORT
RAM
BLOCK
13

FIG. 1B

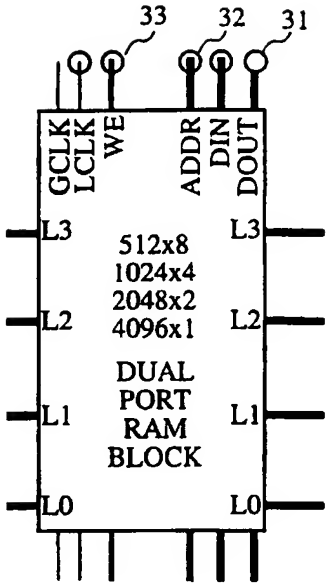

33
32
31
GCLK
LCLK
WE
ADDR
DIN
DOUT
L3
L2
L1
L0
512x8
1024x4
2048x2
4096x1
DUAL
PORT
RAM
BLOCK

FIG. 1C

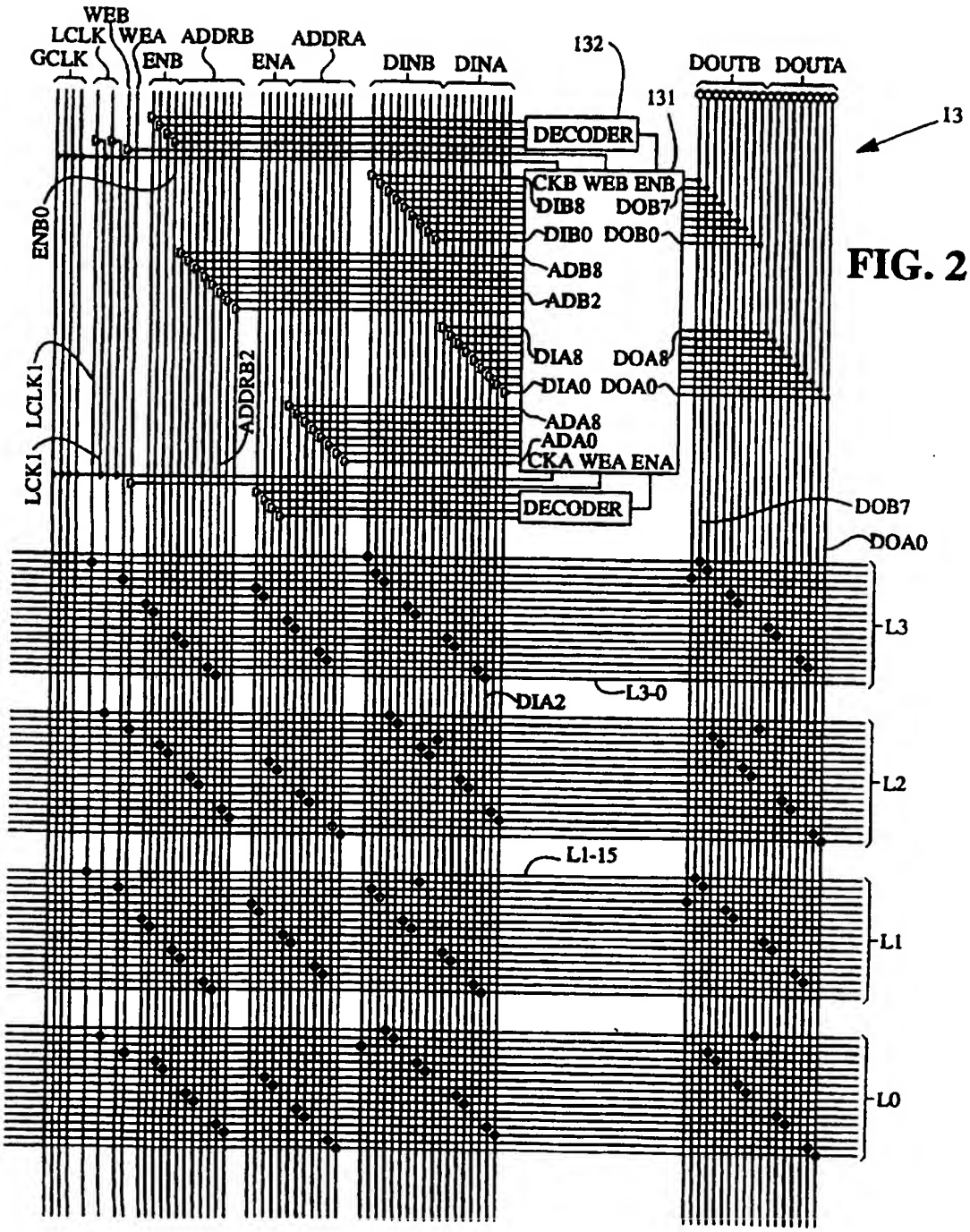

WEB
LCLK
WEA
ADDRB
ADDRA
GCLK
ENB
ENA
DINB
DINA
132
DOUTB
DOUTA
131
13
DECODER
CKB WEB ENB
DIB8
DOB7
DIB0
DOB0
ADB8
ADB2
DIA8
DOA8
DIA0
DOA0
ADA8
ADA0
CKA WEA ENA
DECODER
ENB0
LCLK1
LCK1
ADDRB2
DOB7
DOA0
L3
L3-0
DIA2
L2
L1-15
L1
L0

FIG. 2

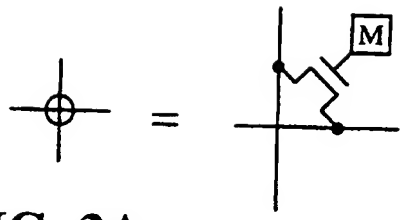


FIG. 3A

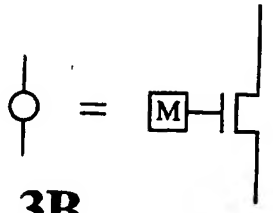


FIG. 3B

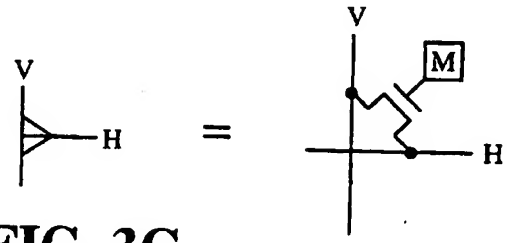


FIG. 3C

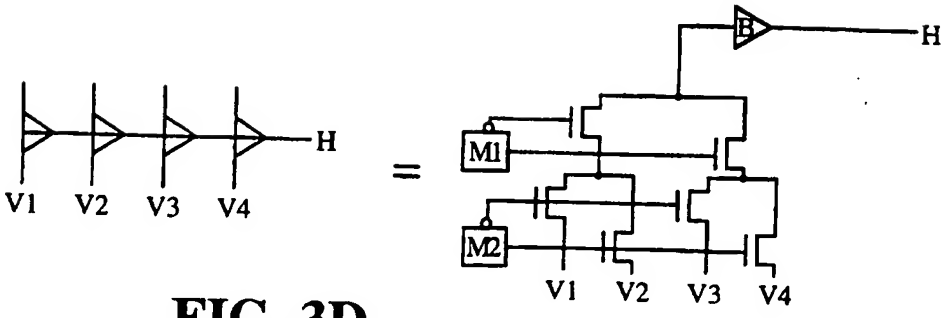


FIG. 3D

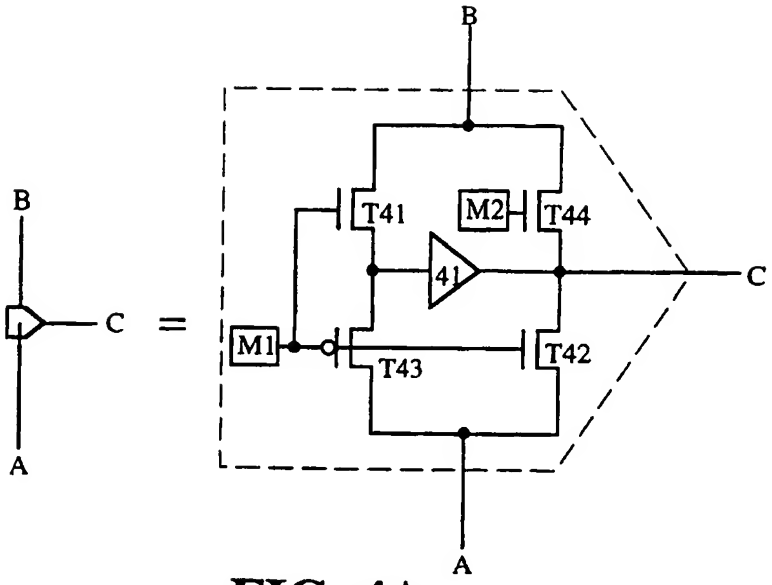


FIG. 4A

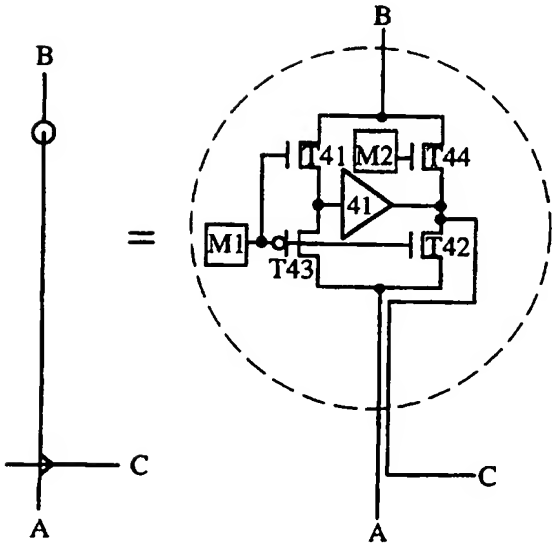


FIG. 4B

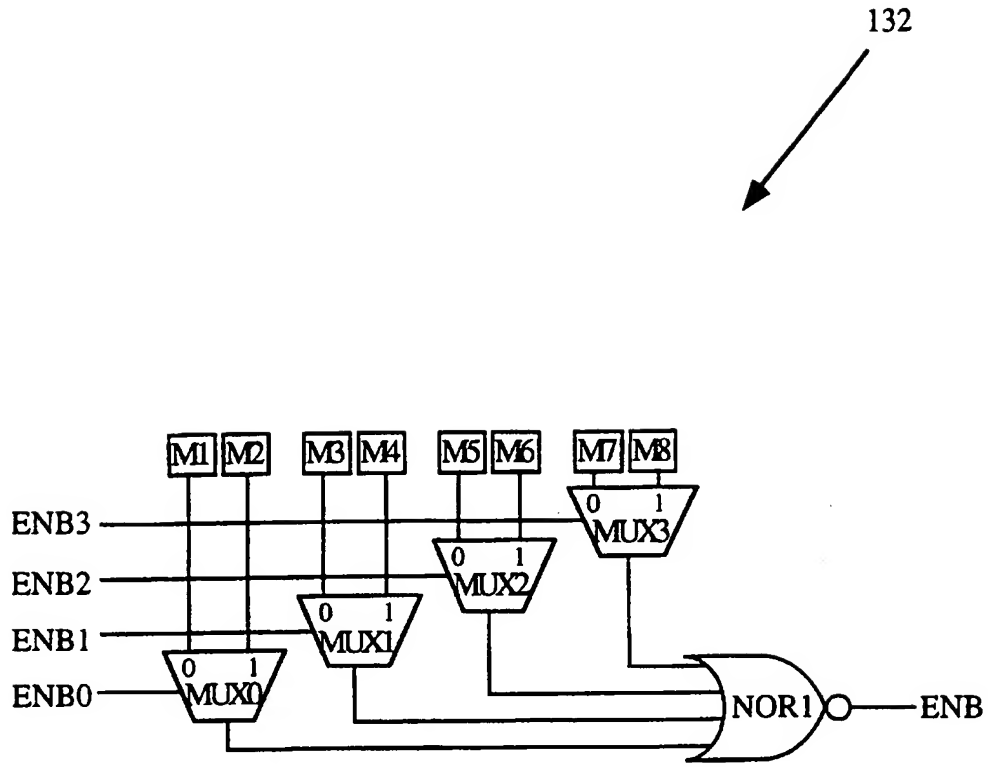

132
M1
M2
M3
M4
M5
M6
M7
M8
ENB3
ENB2
ENB1
ENB0
0
1
MUX3
MUX2
MUX1
MUX0
NOR1
ENB

FIG. 5

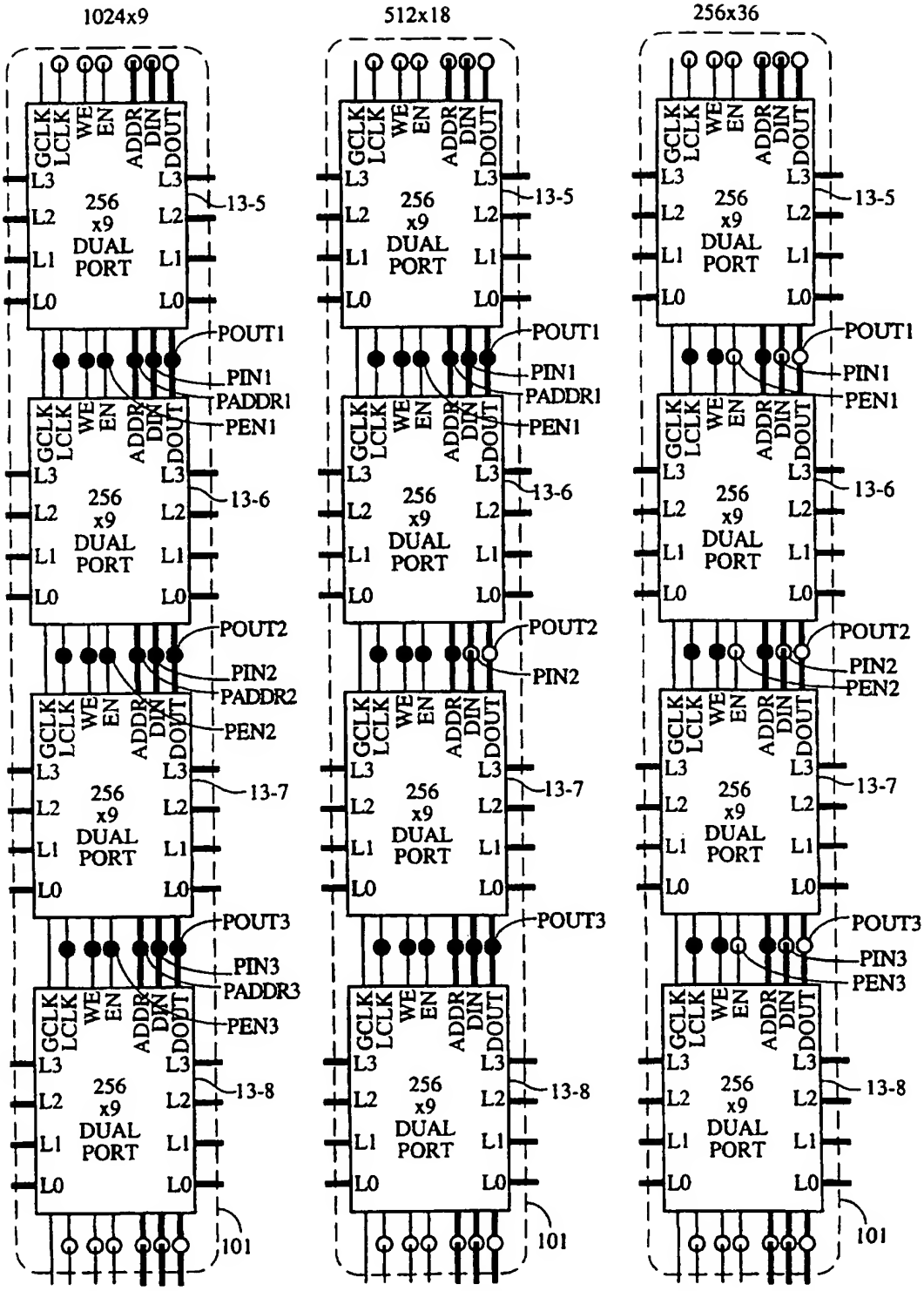


FIG. 6A FIG. 6B FIG. 6C

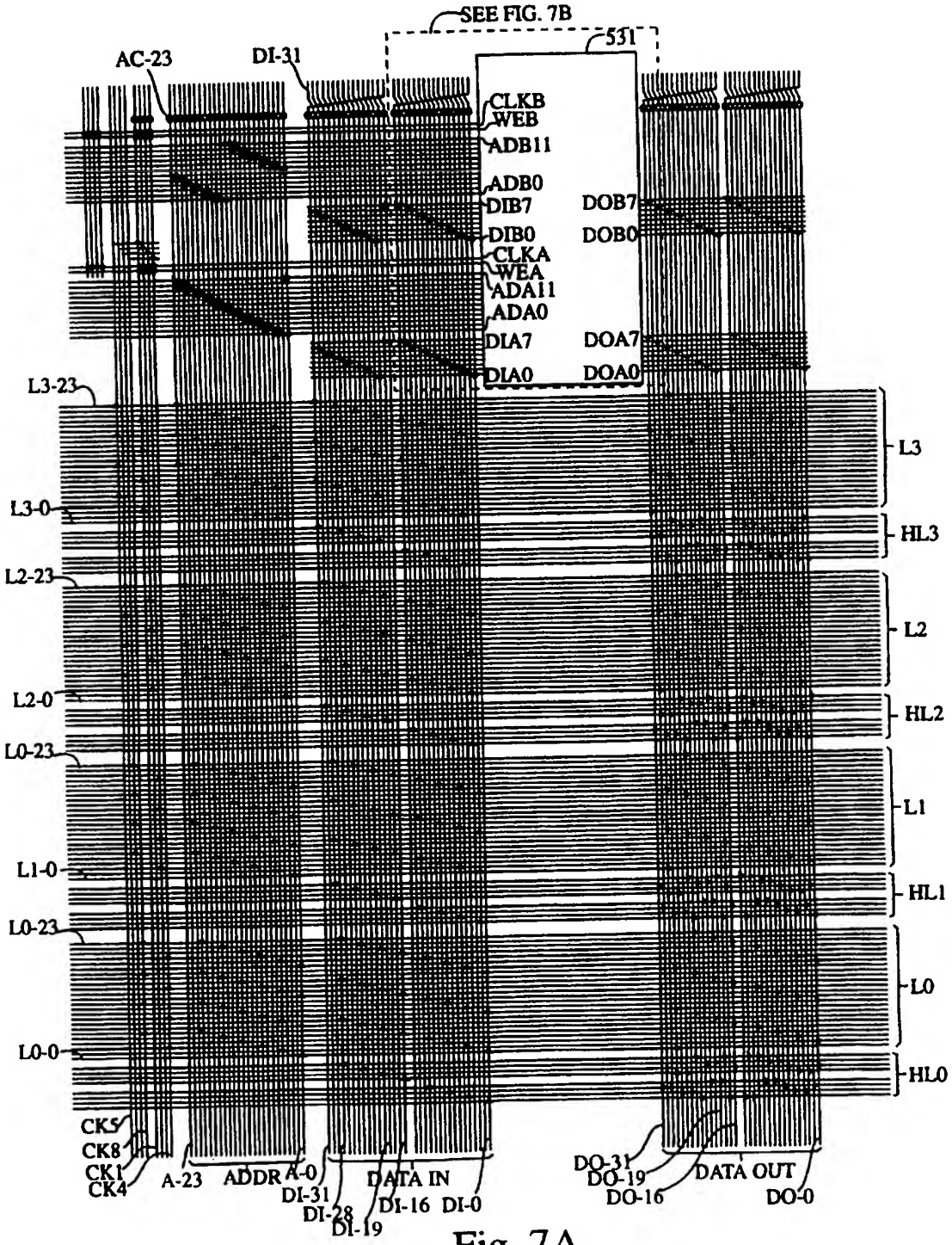

SEE FIG. 7B
531
AC-23
DI-31
CLKB
WEB
ADB11
ADB0
DIB7
DIB0
CLKA
WEA
ADA11
ADA0
DIA7
DIA0
DOB7
DOB0
DOA7
DOA0
L3-23
L3-0
L2-23
L2-0
L0-23
L1-0
L0-23
L0-0
L3
HL3
L2
HL2
L1
HL1
L0
HL0
CK5
CK8
CK1
CK4
A-23
ADDR
A-0
DI-31
DI-28
DI-19
DATA IN
DI-16
DI-0
DO-31
DO-19
DO-16
DATA OUT
DO-0

Fig. 7A

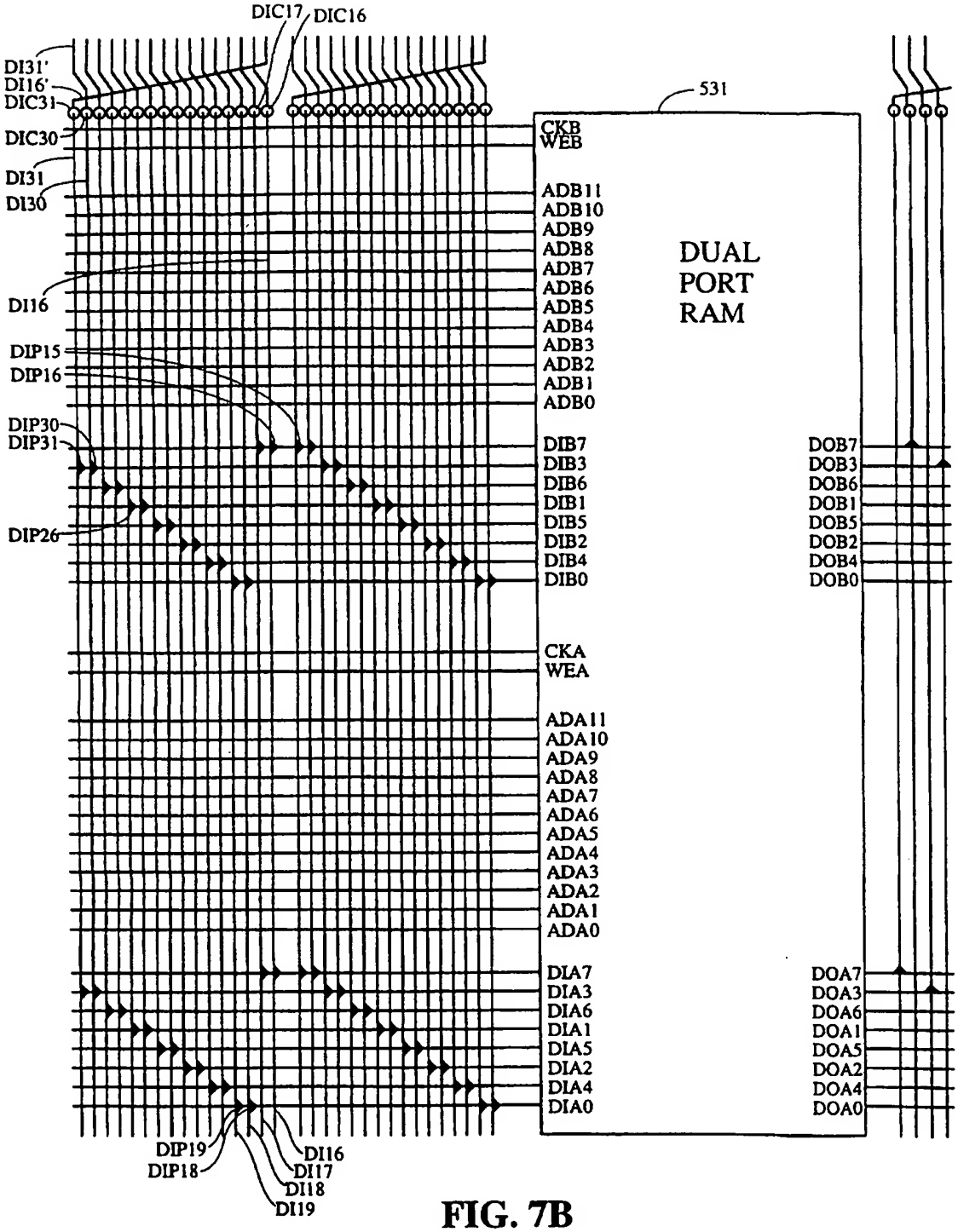

DIC17
DIC16
DI31'
DI16'
DIC31
DIC30
DI31
DI30
DI16
DIP15
DIP16
DIP30
DIP31
DIP26
DIP19
DIP18
DI16
DI17
DI18
DI19
531
CKB
WEB
ADB11
ADB10
ADB9
ADB8
ADB7
ADB6
ADB5
ADB4
ADB3
ADB2
ADB1
ADB0
DUAL
PORT
RAM
DIB7
DIB3
DIB6
DIB1
DIB5
DIB2
DIB4
DIB0
CKA
WEA
ADA11
ADA10
ADA9
ADA8
ADA7
ADA6
ADA5
ADA4
ADA3
ADA2
ADA1
ADA0
DIA7
DIA3
DIA6
DIA1
DIA5
DIA2
DIA4
DIA0
DOB7
DOB3
DOB6
DOB1
DOB5
DOB2
DOB4
DOB0
DOA7
DOA3
DOA6
DOA1
DOA5
DOA2
DOA4
DOA0

FIG. 7B

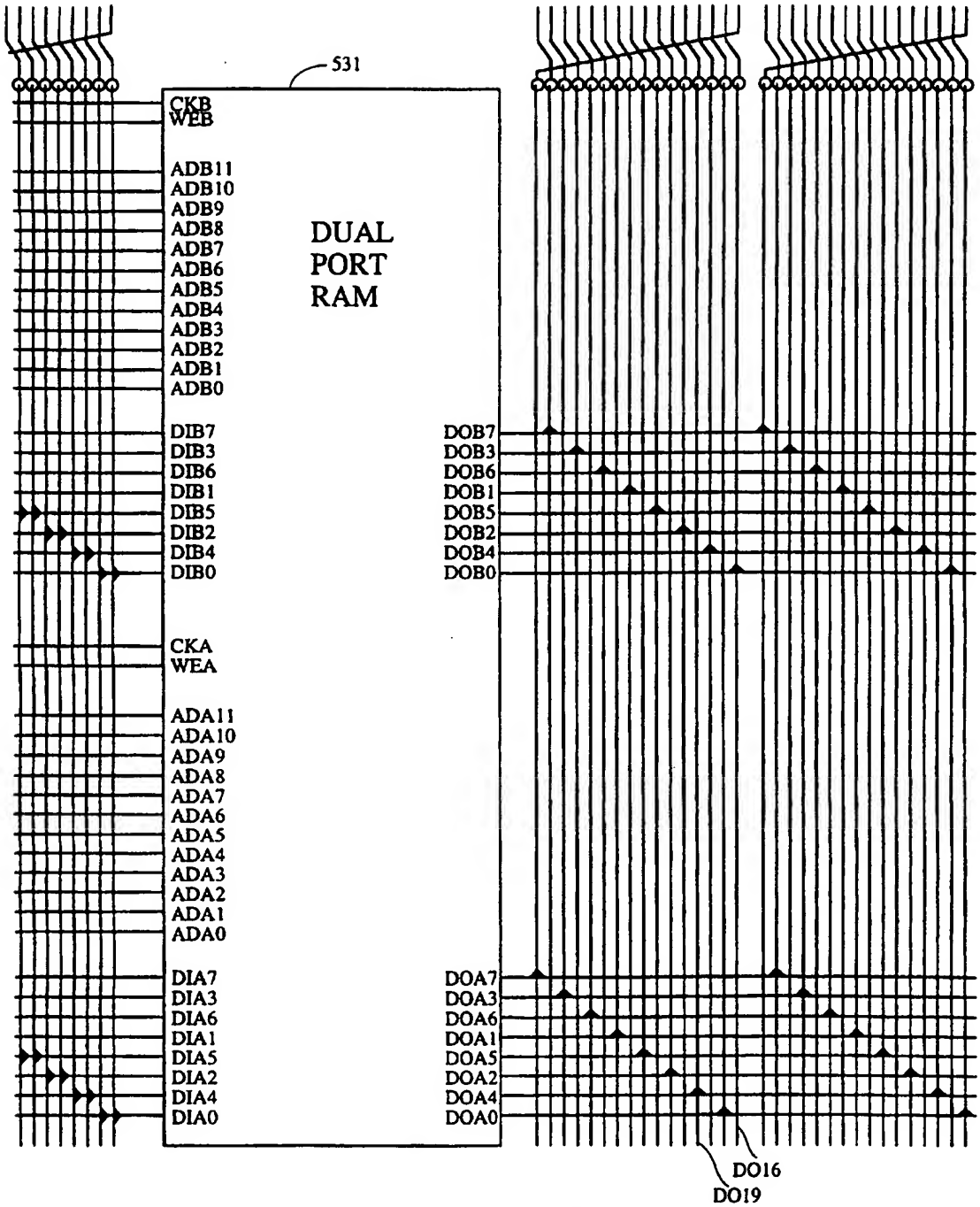

531
CKB
WEB
ADB11
ADB10
ADB9
ADB8
ADB7
ADB6
ADB5
ADB4
ADB3
ADB2
ADB1
ADB0
DUAL
PORT
RAM
DIB7
DIB3
DIB6
DIB1
DIB5
DIB2
DIB4
DIB0
CKA
WEA
ADA11
ADA10
ADA9
ADA8
ADA7
ADA6
ADA5
ADA4
ADA3
ADA2
ADA1
ADA0
DIA7
DIA3
DIA6
DIA1
DIA5
DIA2
DIA4
DIA0
DOB7
DOB3
DOB6
DOB1
DOB5
DOB2
DOB4
DOB0
DOA7
DOA3
DOA6
DOA1
DOA5
DOA2
DOA4
DOA0
DO16
DO19

FIG. 7C

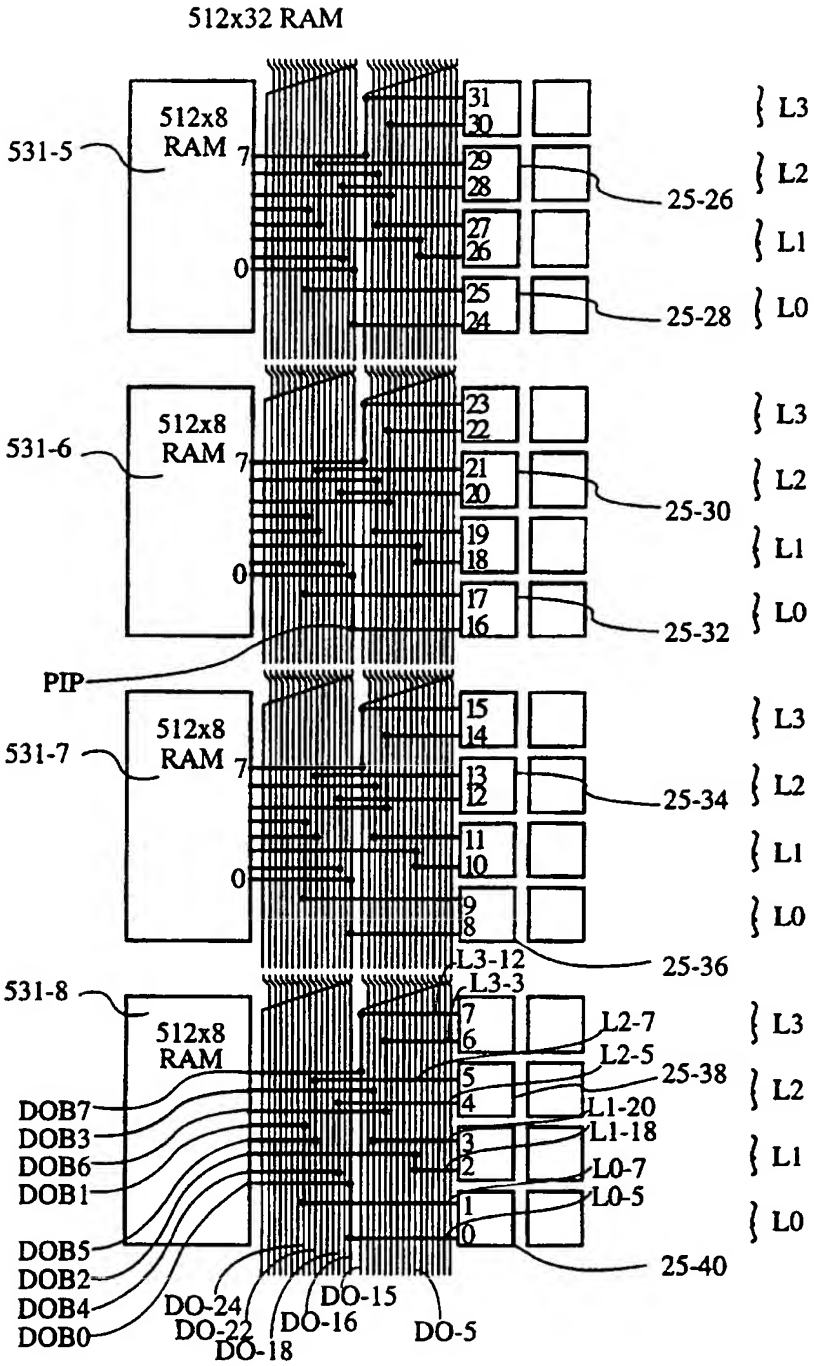

512x32 RAM
512x8 RAM
531-5
531-6
531-7
531-8
PIP
25-26
25-28
25-30
25-32
25-34
25-36
25-38
25-40
L3
L2
L1
L0
L3-12
L3-3
L2-7
L2-5
L1-20
L1-18
L0-7
L0-5
DOB7
DOB3
DOB6
DOB1
DOB5
DOB2
DOB4
DOB0
DO-24
DO-22
DO-18
DO-15
DO-16
DO-5

Fig. 8A

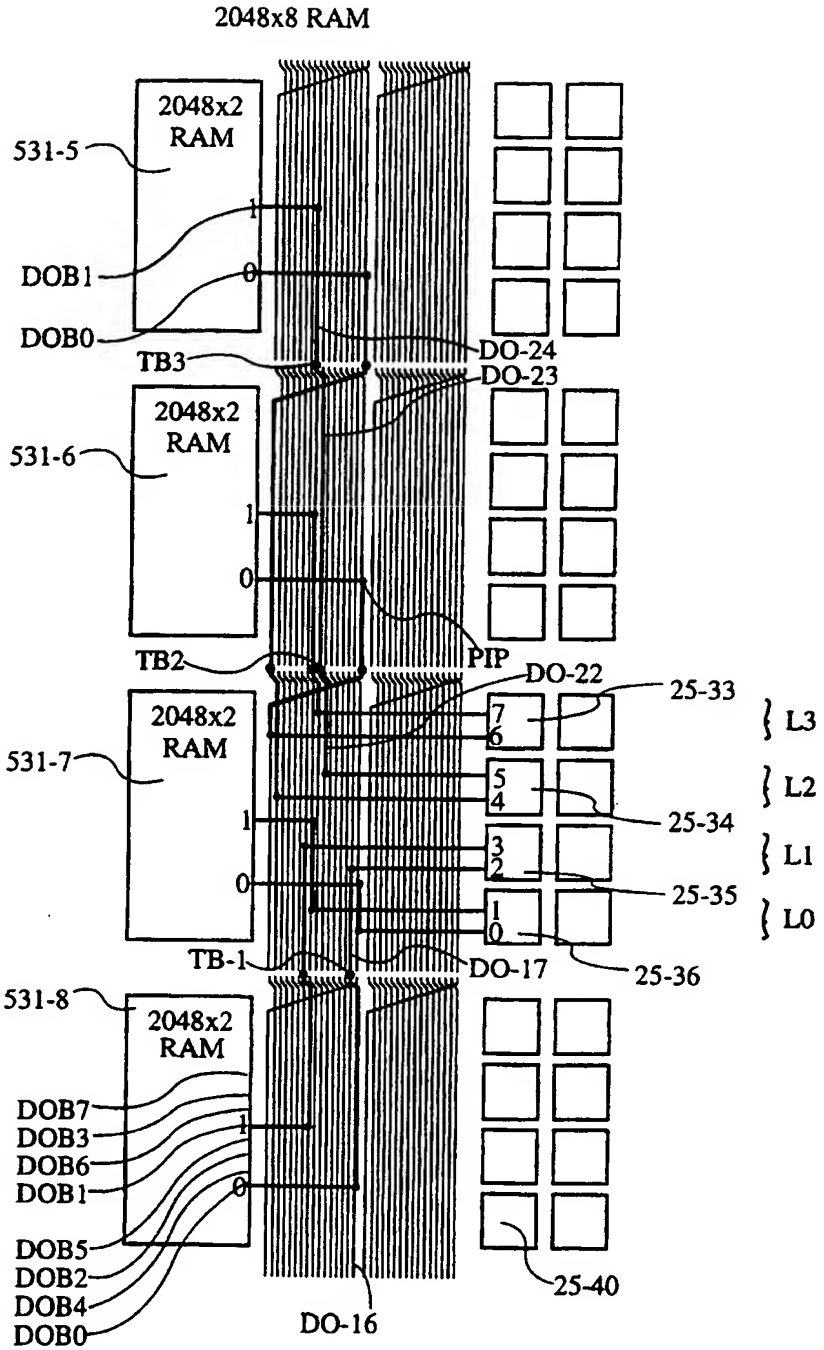

2048x8 RAM
2048x2 RAM
531-5
DOB1
DOB0
TB3
DO-24
DO-23
531-6
TB2
PIP
DO-22
25-33
531-7
25-34
25-35
TB-1
DO-17
25-36
L3
L2
L1
L0
531-8
DOB7
DOB3
DOB6
DOB1
DOB5
DOB2
DOB4
DOB0
DO-16
25-40

Fig. 8B

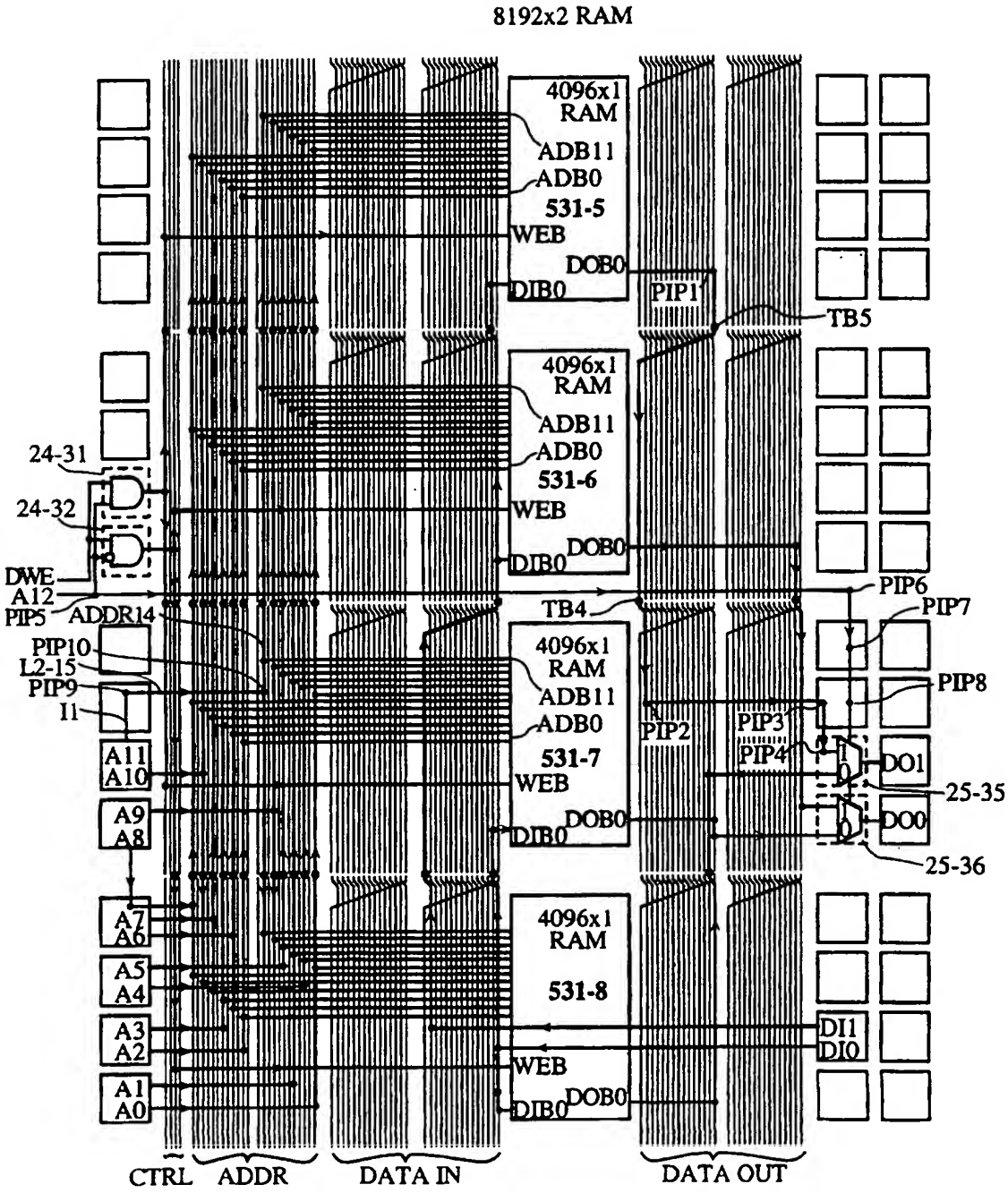

8192x2 RAM
4096x1 RAM
ADB11
ADB0
531-5
WEB
DOB0
DIB0
PIP1
TB5
531-6
531-7
531-8
24-31
24-32
DWE
A12
PIP5
ADDR14
TB4
PIP6
PIP7
PIP10
L2-15
PIP9
I1
PIP8
PIP2
PIP3
PIP4
DO1
DO0
25-35
25-36
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
DI1
DI0
CTRL
ADDR
DATA IN
DATA OUT

Fig. 8C

INTERNATIONAL SEARCH REPORT

Inten .ial Application No
PCT/US 97/10279

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 H03K19/177

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 H03K

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
E	WO 97 30517 A (ACTEL CORP) 21 August 1997 see the whole document ---	1,2,6,7
X	BURSKY D: "EFFICIENT RAM-BASED FPGAS EASE SYSTEM DESIGN" ELECTRONIC DESIGN, vol. 44, no. 2, page 53/54, 58, 60, 62 XP000580106 see page 60, left-hand column, paragraph 2 - page 62, right-hand column, last paragraph; figure 4 --- -/--	1,11

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

2

Date of the actual completion of the international search	Date of mailing of the international search report
9 October 1997	2 0. 10. 97

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Blaas, D-L

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inte. ...nal Application No
PCT/US 97/10279

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	BROWN S ET AL: "FPGA AND CPLD ARCHITECTURES: A TUTORIAL" IEEE DESIGN & TEST OF COMPUTERS, vol. 13, no. 2, pages 42-57, XP000596695 see page 54, left-hand column, last paragraph; figure 23 ---	1
X	US 5 550 782 A (CLIFF RICHARD G ET AL) 27 August 1996 see column 3, line 43 - line 56 see column 15, line 38 - column 24, line 36; figures 20-25 ---	1,7,8,11
A	US 5 517 135 A (YOUNG STEVEN P) 14 May 1996 cited in the application see abstract -----	2-5, 12-15

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inter nal Application No
PCT/US 97/10279

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9730517 A	21-08-97	NONE	
US 5550782 A	27-08-96	US 5436575 A	25-07-95
		US 5260610 A	09-11-93
		US 5260611 A	09-11-93
		GB 2289964 A	06-12-95
		US 5668771 A	16-09-97
		EP 0530985 A	10-03-93
		EP 0786871 A	30-07-97
		JP 6318638 A	15-11-94
		US 5371422 A	06-12-94
		US 5376844 A	27-12-94
		US 5485103 A	16-01-96
		DE 69312563 D	04-09-97
		EP 0569137 A	10-11-93
		EP 0746102 A	04-12-96
		EP 0746103 A	04-12-96
		JP 6053817 A	25-02-94
US 5517135 A	14-05-96	NONE	